Matematica delle Reti Neurali

Volume VII

Silvano Salvador

Grandi Modelli Linguistici · Diffusione · GNN · Meta-apprendimento

Apprendimento Federato · Interpretabilità Meccanicistica · Modelli Efficienti · Teoria Moderna

Sezioni 407–446 — Fondamenti avanzati dell'apprendimento automatico moderno

LLM & Scaling LawsRLHFDiffusion DDPM Score MatchingGNN & WL TestMAML Federated LearningMechanistic Interp.LoRA QuantizzazioneGrokkingLoss Landscape

Tokenizzazione e Vocabolario: la Porta di Accesso al Linguaggio

Come le sequenze di testo diventano sequenze di interi — Byte-Pair Encoding, WordPiece, SentencePiece e l'impatto della tokenizzazione sulla matematica del modello.

407.1 · Il Problema della Tokenizzazione

Qualunque modello che tratti il linguaggio come sequenza di simboli discreta deve affrontare una domanda preliminare: quali sono i simboli? La scelta più ovvia — i caratteri singoli dell'alfabeto — produce sequenze troppo lunghe e perde la struttura morfologica delle parole; la scelta opposta — le parole dell'intero dizionario — produce un vocabolario immenso (l'italiano ha centinaia di migliaia di forme flesse), con problemi di rarà per parole tecniche, nomi propri e neologismi. I metodi moderni di **tokenizzazione sub-parola** trovano un compromesso: le parole comuni vengono rappresentate come un unico token, le parole rare vengono spezzate in sotto-unità morfologicamente significative, e le sequenze arbitrarie di caratteri (codice, URL, formule) possono sempre essere rappresentate, nel caso peggiore, con sequenze di byte singoli.

L'algoritmo **Byte-Pair Encoding (BPE)** (Sennrich, Haddow e Birch 2016, applicato a LLM da Radford et al. per GPT-2) apprende il vocabolario in modo completamente non supervisionato. Si inizia con ogni byte distinto (o ogni carattere) come token atomico, ottenendo un vocabolario di partenza di 256 elementi (nel caso byte-level). Si itera poi il seguente passo: si conta la frequenza di ogni coppia adiacente di token nel corpus di training; si fonde la coppia più frequente in un unico nuovo token; si aggiorna il corpus sostituendo tutte le occorrenze della coppia con il nuovo token. Si ripete fino a raggiungere la dimensione di vocabolario desiderata — tipicamente 32.000 a 100.000 token per i LLM moderni. Il risultato è un vocabolario adattato alla distribuzione del corpus: le parole frequenti ottengono token dedicati, le sequenze rare

vengono decomposta in parti più comuni. La tokenizzazione BPE è *deterministica* dato il vocabolario appreso: ogni stringa ha un'unica tokenizzazione, calcolabile in $O(n)$ con la struttura dati a priorità appropriata.

407.2 · WordPiece, SentencePiece e Effetti sul Modello

Il **WordPiece** (Schuster e Nakamura 2012, adottato da BERT) differisce dal BPE nella funzione di fusione: invece di scegliere la coppia più frequente, sceglie la coppia che massimizza il rapporto tra la verosimiglianza del corpus con la fusione e quella senza. Formalmente, si fonde la coppia (a, b) che massimizza $\log P(ab) - \log P(a) - \log P(b) = \log \frac{P(ab)}{P(a)P(b)}$ — il logaritmo della pointwise mutual information. Questo criterio privilegia coppie che co-occorrono molto più di quanto ci si aspetti per caso, catturando unità morfologicamente coese anche se non frequentissime in assoluto. **SentencePiece** (Kudo e Richardson 2018) elimina la dipendenza dalla segmentazione pre-esistente in parole: opera direttamente sulla sequenza di caratteri Unicode senza assumere che gli spazi separino le parole — una proprietà cruciale per le lingue che non usano spazi (cinese, giapponese, tailandese) o che usano morfologie agglutinanti (finlandese, turco).

La scelta della tokenizzazione non è innocua per la matematica del modello. La **lunghezza media della tokenizzazione** per lingua influenza direttamente quanto un modello può "vedere" all'interno della sua finestra di contesto: l'inglese si tokenizza efficientemente (circa 1 token per 4 caratteri), mentre il greco o l'arabo con diacritici richiedono più token per parola, accorciando effettivamente la finestra contestuale disponibile per tali lingue. Recenti lavori sull'**ortographic tokenization** e sui tokenizzatori appresi congiuntamente al modello (Megabyte, MEGALODON) suggeriscono che la tokenizzazione ottimale dovrebbe dipendere dall'architettura e dal task, non essere fissata a priori — una direzione di ricerca attiva che potrebbe portare a modelli più equi tra lingue diverse.

Pre-training dei Grandi Modelli Linguistici: Obiettivi e Dati

Come i LLM acquisiscono la conoscenza — la modellazione del linguaggio causale, il masked language modeling, e l'impatto della qualità dei dati di pre-training.

408.1 · La Modellazione del Linguaggio Causale

I modelli linguistici autoregressivi come GPT modellano la probabilità di una sequenza di token $(x_1, x_2, \ldots, x_T)$ tramite la regola della catena: $P(x_1, \ldots, x_T) = \prod_{t=1}^{T} P(x_t \mid x_1, \ldots, x_{t-1})$.

Il pre-training minimizza la perdita di **cross-entropy causale** su un corpus di dimensione astronomica:

$$\mathcal{L}_{\text{CLM}}(\theta) = -\frac{1}{T}\sum_{t=1}^{T} \log P_\theta(x_t \mid x_1, \ldots, x_{t-1})$$

Questo obiettivo ha una proprietà notevole: è un segnale di supervisione autosufficiente che non richiede alcuna annotazione umana. Qualunque testo esistente è automaticamente un esempio di training — la risposta corretta è semplicemente il token successivo. La scala dei dati di pre-training dei LLM moderni è difficile da concettualizzare: GPT-3 è stato addestrato su circa 300 miliardi di token, LLaMA-2 su 2 trilioni, Llama-3 su circa 15 trilioni. Il corpus tipico comprende pagine web filtrate (Common Crawl), libri digitalizzati, codice sorgente (GitHub), articoli scientifici (ArXiv), e Wikipedia in decine di lingue. La **qualità dei dati** è emersa come fattore dominante sulla quantità: il lavoro Phi-1 (Gunasekar et al. 2023) ha mostrato che un modello da 1 miliardo di parametri addestrato su 7 miliardi di token di "textbook quality" supera modelli molto più grandi addestrati su dati non filtrati, suggerendo che la cura del corpus di pre-training — deduplicazione, filtraggio della qualità, bilanciamento tra domini — ha un impatto proporzionalmente maggiore dell'aumento bruto di parametri o token.

408.2 · Masked Language Modeling e Obiettivi Bidirezionali

BERT (Devlin et al., 2018) introduce il **Masked Language Modeling (MLM)**: dato un testo, il 15% dei token viene sostituito casualmente con un token speciale `[MASK]`, e il modello deve predire i token originali condizionatamente al contesto bidirezionale. L'obiettivo MLM non è una densità autoregressiva completa ma un obiettivo di denoise pesudo-marginale: per ogni posizione mascherata i la loss è $-\log p_\theta(x_i \mid \mathbf{x}_{\setminus i})$ dove $\mathbf{x}_{\setminus i}$ indica la sequenza con la posizione i mascherata. La rappresentazione risultante cattura contesto sinistro e destro simultaneamente, producendo embedding contestuali superiori per task di comprensione. Varianti successive (RoBERTa, DeBERTa) ottimizzano la procedura di masking: RoBERTa rimuove il task ausiliario NSP (Next Sentence Prediction) e usa masking dinamico rigenerato ad ogni epoca; DeBERTa aggiunge **Disentangled Attention** separando content embedding e position embedding in matrici distinte, migliorando la sensibilità alla posizione relativa dei token.

Un obiettivo ibrido tra CLM e MLM è **XLNet** (Yang et al., 2019), che usa permutation language modeling: il modello massimizza la verosimiglianza su tutte le $T!$ permutazioni dell'ordine dei token, ma la rete mantiene l'ordine posizionale originale — viene così modellata la distribuzione congiunta completa senza bisogno di token [MASK] artificiali. Il limite computazionale delle

permutazioni viene gestito campionando un sottoinsieme e mantenendo solo le ultime K posizioni della permutazione come target. La famiglia T5 (Raffel et al., 2020) unifica tutti i task NLP in un framework testo-a-testo con **span corruption**: sequenze di token contigui vengono sostituite da un token sentinella, e il decoder deve ricostruire i span eliminati — obiettivo più denso e informativo del semplice token masking.

408.3 · Dati di Pre-training: Qualità, Scala e Bias

La qualità e la composizione del corpus di pre-training ha impatto critico sulle capacità emergenti del modello. I dataset moderni come **The Pile** (Gao et al., 2020), **ROOTS** e **RedPajama** combinano Common Crawl filtrato, Wikipedia, libri digitalizzati (Project Gutenberg, Books3), codice sorgente (GitHub), articoli accademici (ArXiv, PubMed) e dati conversazionali. Il filtraggio della qualità usa euristiche (lunghezza minima, rapporto simboli/parole, rimozione di contenuti duplicati con MinHash LSH) e classificatori addestrati su testi curati come Wikipedia come positivi e dati Common Crawl non filtrati come negativi. I **token repetition penalty** e il deduplication a livello di n-grammi (Lee et al., 2022) migliorano la generalizzazione riducendo la memorizzazione di frammenti frequenti. La composizione del corpus introduce bias sistematici: sovra-rappresentazione dell'inglese e delle lingue europee, bias di genere e razza presenti nei testi web, conoscenza enciclopedica concentrata nel dominio occidentale. Le leggi di scala di Hoffmann et al. (2022) indicano che il budget ottimale alloca metà FLOP ai parametri e metà ai token — un modello con N parametri dovrebbe essere addestrato su $\approx 20N$ token per massimizzare l'efficienza computazionale.

BERT (Devlin et al. 2019) ha introdotto il **Masked Language Modeling (MLM)** come alternativa bidirezionale alla modellazione causale: si mascherano casualmente il 15% dei token di input (sostituendoli con il token speciale [MASK], un token casuale, o lasciandoli invariati in proporzioni 80/10/10), e il modello deve predire i token originali nelle posizioni mascherate dato il contesto completo sia sinistro che destro. Il vantaggio del MLM è la bidirezionalità: ogni token può influenzare e essere influenzato da tutti gli altri token nella sequenza, producendo rappresentazioni contestuali più ricche rispetto alle rappresentazioni causali che vedono solo il contesto sinistro. Il prezzo è l'impossibilità di usare direttamente il modello per la generazione autoregressiva: BERT non può generare testo token per token perché i suoi parametri dipendono dall'intera sequenza. Modelli ibridi come BART (Lewis et al. 2020) e T5 (Raffel et al. 2020) combinano un encoder bidirezionale con un decoder causale — adatti sia alla comprensione che alla generazione — mentre UniLM (Dong et al. 2019) usa maschere di attenzione speciali per alternare tra modalità causale e bidirezionale all'interno dello stesso modello.

Scaling Laws: la Matematica della Crescita dei LLM

Come performance, dati e parametri si relazionano quantitativamente — le leggi di potenza di Kaplan e Hoffmann, il compute-optimal training e le implicazioni pratiche.

409.1 · Le Leggi di Scala di Kaplan et al.

Nel 2020, Kaplan, McCandlish, Henighan e colleghi di OpenAI hanno pubblicato un risultato empirico che ha cambiato il modo in cui i laboratori di IA pianificano i propri esperimenti: la perdita di cross-entropy di un modello linguistico segue **leggi di potenza** nel numero di parametri N, nella dimensione del dataset D e nel budget computazionale C (misurato in FLOP), con scarsa dipendenza dall'architettura specifica (numero di layer vs. dimensione nascosta). Le relazioni fondamentali sono:

$$L(N) \approx \left(\frac{N_c}{N}\right)^{\alpha_N}, \quad L(D) \approx \left(\frac{D_c}{D}\right)^{\alpha_D}, \quad L(C) \approx \left(\frac{C_c}{C}\right)^{\alpha_C}$$

con esponenti $\alpha_N \approx 0.076$, $\alpha_D \approx 0.095$, $\alpha_C \approx 0.050$ per i transformer autoregressivi su linguaggio. La piccola magnitudine di questi esponenti implica che raddoppiare il budget computazionale riduce la perdita solo del 3%; occorre aumentare il compute di un fattore 10 per ridurre la perdita del 12%. Questo è al tempo stesso scoraggiante (il progresso è lento) e illuminante (il progresso è *prevedibile*): le leggi di scala permettono di stimare quantitativamente le prestazioni di un modello da 10 miliardi di parametri addestrando solo modelli da 100 milioni, estrapolando lungo la curva di potenza. Le leggi di scala hanno anche implicazioni sull'**allocazione ottimale del budget**: fissato un budget computazionale C, quale combinazione di N (parametri) e D (token di training) massimizza le prestazioni? Kaplan et al. suggerivano di aumentare N molto più rapidamente di D, portando a modelli come GPT-3 (175B parametri, 300B token) largamente sotto-addestrati rispetto all'ottimo compute-efficient.

409.2 · Chinchilla e il Training Compute-Optimal

Hoffmann, Borgeaud e colleghi di DeepMind (2022) hanno sfidato la ricetta di Kaplan con un esperimento sistematico che ha addestrato oltre 400 modelli su un range di dimensioni (70M–16B parametri) e budget di token (5B–500B), arrivando alla conclusione che l'allocazione ottimale rispetta un rapporto molto più bilanciato: il numero di token di training dovrebbe crescere *linearmente* con il numero di parametri, $D^* \approx 20N$. Il modello **Chinchilla** da 70

miliardi di parametri addestrato su 1,4 trilioni di token ha superato Gopher (280B parametri, stesso budget computazionale) su quasi tutti i benchmark, dimostrando che la maggior parte dei modelli precedenti era drammaticamente "sotto-addestrata" rispetto all'ottimo compute-efficient. La regola di Chinchilla ha trasformato la pratica del settore: LLaMA, Mistral, e quasi tutti i modelli open-source successivi al 2022 sono dimensionati sulla base di questa regola, spesso con ancora più token di quanto Chinchilla suggerisca ($D = 100N$ o più) per massimizzare le prestazioni durante l'inferenza a budget fisso. Il trade-off sottostante è tra training cost (più dati = più FLOP durante il training) e inference cost (modello più piccolo = meno FLOP per token generato): le organizzazioni che devono servire il modello a miliardi di utenti preferiscono modelli più piccoli ma più addestrati, anche se il training compute è più elevato.

Fine-tuning Supervisionato e Instruction Tuning

Trasformare un modello di linguaggio grezzo in un assistente utile — l'SFT con dati dimostrativi, il format delle istruzioni e il problema del catastrophic forgetting.

410.1 · Dalla Completamento del Testo al Seguire Istruzioni

Un LLM pre-addestrato minimizza la perdita di prossimo-token su testo di internet — un obiettivo che lo rende abile a completare qualunque stile di scrittura ma non a seguire istruzioni nel formato conversazionale che gli utenti si aspettano. Il **Supervised Fine-Tuning (SFT)** adatta il modello a questo formato usando dimostrazioni curate da umani: ogni esempio è una coppia (istruzione, risposta desiderata) in un formato specifico, e il modello viene addestrato con cross-entropy causale solo sui token della risposta — non dell'istruzione, che funge da contesto. La perdita è quindi $\mathcal{L}_{\text{SFT}} = -\sum_{t \in \text{risposta}} \log P_\theta(x_t \mid x_{lt;t})$, con il gradiente azzerato sulle posizioni dell'istruzione tramite una maschera. La dimensione del dataset SFT è sorprendentemente piccola: InstructGPT (Ouyang et al. 2022) usa circa 13.000 dimostrazioni umane, e il paper Alpaca dimostra che 52.000 istruzioni generate da GPT-4 possono produrre comportamenti da assistente paragonabili. Questo effetto di *amplificazione* del SFT — pochi dati di alta qualità trasformano radicalmente il comportamento del modello — riflette il fatto che le capacità sottostanti (ragionamento, conoscenza, comprensione) erano già presenti nel modello pre-addestrato; il SFT insegna essenzialmente il *formato* e il *registro* appropriati, non nuova conoscenza.

410.2 · Catastrophic Forgetting e Tecniche di Mitigazione

Il fine-tuning su un dataset piccolo e specializzato può compromettere le capacità generali del modello pre-addestrato — il fenomeno del **catastrophic forgetting**: i parametri vengono modificati per ottimizzare il nuovo obiettivo, sovrascrivendo parte della conoscenza acquisita durante il pre-training. Le tecniche principali di mitigazione sono tre. La prima è il **learning rate basso** con scheduler conservativo (warmup + cosine decay a 10^{-5}–10^{-4}) che limita la magnitudine degli aggiornamenti. La seconda è la **regolarizzazione Elastic Weight Consolidation (EWC)** (Kirkpatrick et al. 2017): si penalizza le modifiche ai parametri importanti per il task precedente, pesando la loss di fine-tuning con $\frac{\lambda}{2}\sum_i F_i(\theta_i - \theta_i^*)^2$ dove F_i è la diagonale della matrice di Fisher del pre-training e θ^* sono i pesi pre-addestrati. La terza, sempre più dominante nella pratica, è l'uso di metodi **parameter-efficient** come LoRA (§440): modificare solo una piccola frazione dei parametri limita strutturalmente l'entità del forgetting preservando la maggior parte della rappresentazione pre-addestrata.

410.3 · Instruction Tuning Multi-task e il Ruolo dei Template

L'**instruction tuning** trasforma un modello di completamento in un assistente generalista addestrandolo su raccolte di task eterogenei descritti in linguaggio naturale. FLAN (Wei et al., 2021) dimostra che addestrare su più di 60 task con template diversificati migliora la zero-shot generalization su task non visti: il modello apprende la struttura generale del seguire istruzioni, non solo le specifiche di ogni singolo task. La costruzione dei dataset di istruzioni richiede attenzione alla diversità dei template — ogni task viene espresso con 10 o più formulazioni diverse per evitare che il modello si adatti alla struttura sintattica piuttosto che al significato semantico. Self-Instruct (Wang et al., 2022) genera automaticamente nuove istruzioni usando il modello stesso come generatore, guidato da un piccolo seme di istruzioni scritte a mano: un ciclo di generazione-filtraggio-aggiunta produce dataset di decine di migliaia di esempi con costo minimo. Alpaca (Taori et al., 2023) adatta questo approccio a LLaMA generando 52k istruzioni via GPT-3.5, dimostrando che modelli da 7B parametri instruction-tuned competono con GPT-3 su molti benchmark. La qualità dei dati domina la quantità: Dolly (Conover et al., 2023) con soli 15k esempi scritti da umani produce comportamenti istruttivi più coerenti di dataset automatici dieci volte più grandi.

Il **formato di conversazione multi-turno** — usato da OpenAI, Anthropic e Meta con ChatML, HH e Llama-2-chat — permette al modello di mantenere il contesto su più scambi, fondamentale per applicazioni di assistenza iterativa. La scelta degli iperparametri durante il fine-tuning è critica: learning rate troppo alti distruggono le rappresentazioni pre-addestrate, mentre learning rate troppo bassi non producono adattamento sufficiente. La pratica comune

usa learning rate 10–100 volte inferiori a quelli del pre-training, warm-up breve (100–500 passi) e cosine decay. Il numero di epoch è piccolo (1–3) perché con dataset di istruzioni tipici il modello overfita rapidamente — un singolo passaggio sui dati spesso è sufficiente se il dataset è sufficientemente diversificato e bilanciato per tipo di task.

RLHF e Constitutional AI: Allineare i Modelli alle Preferenze Umane

Il reinforcement learning dal feedback umano — il reward model, PPO applicato agli LLM, e l'approccio Constitutional AI di Anthropic.

411.1 · La Pipeline RLHF

Il **Reinforcement Learning from Human Feedback (RLHF)** (Christiano et al. 2017, applicato agli LLM da Ouyang et al. 2022 per InstructGPT) è la tecnica che ha trasformato i LLM da completatori di testo a assistenti conversazionali affidabili. La pipeline si articola in tre fasi successive. Nella prima fase si raccoglie un dataset di **comparazioni umane**: a valutatoti umani vengono mostrate due o più risposte generate dal modello alla stessa domanda, e scelgono quale è migliore secondo criteri di utilità, onestà e sicurezza. Nella seconda fase si addestra un **reward model (RM)** a predire quale risposta gli umani preferirebbero: dato il prompt e la risposta, il RM produce uno scalare $r_\phi(\mathbf{x}, \mathbf{y})$ addestrato con la loss Bradley-Terry su coppie comparate: $\mathcal{L}_{\text{RM}} = -\mathbb{E}_{(\mathbf{x},\mathbf{y}_w,\mathbf{y}_l)\sim\mathcal{D}}[\log \sigma(r_\phi(\mathbf{x},\mathbf{y}_w) - r_\phi(\mathbf{x},\mathbf{y}_l))]$ dove $\mathbf{y}_w$ è la risposta preferita e $\mathbf{y}_l$ quella meno preferita. Nella terza fase si usa il RM come funzione di reward per ottimizzare il LLM tramite PPO, con una penalizzazione KL che impedisce al modello di allontanarsi troppo dal punto di partenza SFT:

$$\mathcal{L}_{\text{RLHF}}(\theta) = \mathbb{E}_{\mathbf{x}\sim\mathcal{D},\,\mathbf{y}\sim\pi_\theta(\cdot|\mathbf{x})}[r_\phi(\mathbf{x},\mathbf{y}) - \beta\,\text{KL}(\pi_\theta(\cdot|\mathbf{x})\|\pi_{\text{ref}}(\cdot|\mathbf{x}))]$$

La penalizzazione KL è fondamentale: senza di essa, l'ottimizzazione PPO tenderebbe a sfruttare le debolezze del reward model — producendo risposte che ricevono alto score dal RM ma che gli umani troverebbero banali o incoerenti, il fenomeno noto come **reward hacking.** Il coefficiente β bilancia l'ottimizzazione del reward con la preservazione della distribuzione SFT: valori alti rendono il training conservativo ma stabile; valori bassi permettono più ottimizzazione ma a rischio di degenerazione.

411.2 · Direct Preference Optimization: Eliminare il Reward Model

Il **Direct Preference Optimization (DPO)** (Rafailov et al. 2023) ha semplificato radicalmente la pipeline RLHF dimostrando che l'ottimizzazione del reward con penalizzazione KL ha una soluzione analitica chiusa che si può usare direttamente come obiettivo di training supervisionato. Il punto di partenza è la distribuzione ottimale sotto la penalizzazione KL, che si può scrivere come $\pi^*(\mathbf{y}|\mathbf{x}) = \pi_{\text{ref}}(\mathbf{y}|\mathbf{x}) \exp(r(\mathbf{x},\mathbf{y})/\beta)/Z(\mathbf{x})$ dove $Z(\mathbf{x})$ è la costante di normalizzazione. Invertendo questa relazione si ottiene che il reward ottimale è $r^*(\mathbf{x},\mathbf{y}) = \beta \log \frac{\pi^*(\mathbf{y}|\mathbf{x})}{\pi_{\text{ref}}(\mathbf{y}|\mathbf{x})} + \beta \log Z(\mathbf{x})$. Sostituendo nell'obiettivo Bradley-Terry, la costante di normalizzazione si cancella, e si ottiene la loss DPO:

$$\mathcal{L}_{\text{DPO}}(\pi_\theta) = -\mathbb{E}\left[\log \sigma\left(\beta \log \frac{\pi_\theta(\mathbf{y}_w|\mathbf{x})}{\pi_{\text{ref}}(\mathbf{y}_w|\mathbf{x})} - \beta \log \frac{\pi_\theta(\mathbf{y}_l|\mathbf{x})}{\pi_{\text{ref}}(\mathbf{y}_l|\mathbf{x})}\right)\right]$$

DPO ottimizza quindi direttamente le probabilità del modello LLM sulle preferenze comparate, senza addestrare un reward model separato e senza le instabilità numeriche di PPO su sequenze lunghe. La sua semplicità implementativa — una singola loss di cross-entropy modificata — lo ha reso lo standard de facto per l'allineamento nell'open-source, usato in quasi tutti i modelli Llama fine-tuned rilasciati dal 2023.

411.3 · Constitutional AI di Anthropic

Il **Constitutional AI (CAI)** (Bai et al. 2022, Anthropic) affronta l'allineamento con un approccio diverso: invece di dipendere interamente dal giudizio umano per definire i criteri di qualità (laboriosa e costosa raccolta di preferenze), si specifica una "costituzione" — un insieme di principi scritti in linguaggio naturale che definiscono il comportamento desiderato — e si usa il modello stesso per auto-critica e revisione delle proprie risposte rispetto a questi principi. La pipeline CAI alterna due fasi: nella fase *Critique*, il modello valuta la propria risposta rispetto a un sottoinsieme campionato dei principi costituzionali ("questa risposta è rispettosa della dignità umana?", "questa risposta potrebbe essere usata per causare danni?"); nella fase *Revision*, il modello rivisa la risposta incorporando la critica. Questo processo iterativo di auto-revisione guidata dalla costituzione produce dataset sintetici di alta qualità che alimentano un SFT, cui segue RLHF con un reward model addestrato anch'esso tramite valutazioni generate dal modello ("RL from AI Feedback", RLAIF). CAI ha il vantaggio di rendere espliciti e modificabili i criteri di allineamento — la costituzione è un documento leggibile — e di scalare il feedback senza dipendere linearmente dal numero di annotatori umani.

In-Context Learning e Chain-of-Thought Reasoning

Apprendimento senza aggiornamento dei pesi — la matematica dell'ICL, il prompting e la comprensione teorica del ragionamento a catena di pensiero.

412.1 · In-Context Learning: Apprendere dalla Finestra di Contesto

Una delle proprietà più sorprendenti dei LLM di grandi dimensioni è la capacità di **in-context learning (ICL)**: fornendo nella finestra di contesto alcuni esempi di input-output (few-shot prompting), il modello adatta il suo comportamento senza nessun aggiornamento dei pesi. Formalmente, dato un insieme di esempi dimostrativi $\{(x_1, y_1), \ldots, (x_k, y_k)\}$ e una nuova query x_{test}, il modello produce $y_{\text{test}} = \arg\max_y P_\theta(y \mid x_1, y_1, \ldots, x_k, y_k, x_{\text{test}})$ — la predizione ottimale sotto la distribuzione condizionata all'intero contesto. Questa capacità è emergente: non è presente nei modelli piccoli e appare quasi discontinuamente al superamento di una certa scala di parametri. La spiegazione teorica più convincente è quella di Akyurek et al. (2022) e Von Oswald et al. (2023): i layer di attenzione dei transformer implementano implicitamente un algoritmo di gradient descent nel contesto — ogni passo di attenzione corrisponde a un passo di discesa del gradiente sul task implicato dagli esempi dimostrativi, con le chiavi e i valori che fungono da "pesi" e le query da "segnale di aggiornamento". Questa interpretazione spiega perché l'ICL migliora con il numero di esempi (più passi di gradient descent impliciti) e perché è sensibile all'ordine degli esempi (diversa traiettoria di ottimizzazione).

412.2 · Chain-of-Thought: Ragionamento per Passi Intermedi

Il **Chain-of-Thought (CoT) prompting** (Wei et al. 2022, Google Brain) ha dimostrato che fornire esempi di ragionamento passo-passo — "Let's think step by step" — invece di solo esempi input-output aumenta drammaticamente le capacità di ragionamento dei LLM su problemi matematici, logici e di buon senso. L'ipotesi meccanicistica è che i token intermedi del ragionamento fungano da "scratch pad" computazionale: ogni token generato può condizionare i token successivi, effettivamente aumentando la profondità computazionale della "rete" effettiva da $O(L)$ a $O(L \cdot T_{\text{chain}})$ dove L è il numero di layer e T_{chain} il numero di token nella catena di ragionamento. Wei et al. formalizzano questo con l'analogia del circuito: un transformer senza CoT è equivalente a un circuito booleano di profondità $O(L)$, insufficiente a calcolare certi problemi che richiedono profondità $\omega(L)$; con CoT, la profondità effettiva diventa illimitata. Il **Self-Consistency** (Wang et al. 2022) estende il CoT campionando più catene di ragionamento diverse dalla stessa query e scegliendo la risposta finale per maggioranza: la diversità

delle traiettorie di ragionamento funge da ensemble implicito, riducendo la varianza delle predizioni. Recentemente, tecniche come **Scratchpad Training** e **Process Reward Models** (che assegnano reward a ogni passo intermedio invece che solo alla risposta finale) hanno portato i modelli a eccellere su problemi di matematica olimpionica — un benchmark che richiedeva dimostrazione passo-passo verificata da un supervisore separato.

412.3 · Limiti e Meccanismi Interni dell'In-Context Learning

Nonostante le straordinarie capacità dimostrate, l'in-context learning presenta limitazioni strutturali importanti. La **lunghezza della finestra di contesto** vincola il numero di esempi utilizzabili: anche con context window da 128k token, la degradazione delle prestazioni sugli esempi distanti (lost-in-the-middle, Liu et al., 2023) riduce l'efficacia pratica. La sensibilità all'ordine e alla formulazione degli esempi è documentata sperimentalmente: permutare l'ordine degli stessi esempi può variare l'accuratezza di ± 20 punti percentuali su alcuni benchmark, suggerendo che il modello non apprende una vera induzione algoritmica ma piuttosto recupera pattern distribuito nei pesi pre-addestrati. Dal punto di vista meccanicistico, Garg et al. (2022) mostrano che Transformer addestrati da zero su sequenze di regressione lineare imparano a implementare internamente una forma di gradient descent: gli attention head nei layer superiori calcolano una forma di least-squares in-context, suggerendo che il pre-training su dati linguistici instilla algoritmi di apprendimento impliciti nei pesi del modello. Il **Retrieval-Augmented Generation (RAG)** estende l'ICL con un meccanismo di recupero: invece di includere esempi statici nel contesto, un retriever denso (come DPR o ColBERT) recupera dinamicamente i documenti più rilevanti per ogni query da una base di conoscenza esterna — combinando i benefici della memoria parametrica del modello con la precisione e l'aggiornabilità di un database vettoriale.lass="div">

Allineamento e Sicurezza nei Grandi Modelli Linguistici

I problemi fondamentali dell'allineamento — la specifica del goal, la robustezza alla distribuzione, il jailbreaking e le difese, e il dibattito sull'allineamento a lungo termine.

413.1 · Il Problema della Specifica del Goal

Il problema dell'allineamento nei LLM ha due livelli distinti. Il livello *tecnico* riguarda come far sì che il modello faccia ciò che intendiamo: la specifica del reward tramite dimostrazioni umane e preferenze è necessariamente incompleta e può essere sfruttata dal modello. Il problema della **specifica incompleta dell'obiettivo** è formalmente equivalente al "problem of induction": qualunque quantità finita di dati di feedback è compatibile con infinite funzioni di reward,

alcune delle quali producono comportamenti desiderati e altre no. La **Goodhart's Law** descrive il collasso pratico: "quando una misura diventa un obiettivo, cessa di essere una buona misura" — i LLM ottimizzati tramite RLHF tendono a produrre risposte che massimizzano il reward model sfruttandone le debolezze piuttosto che rispondendo genuinamente alle preferenze umane sottostanti. Il livello *strutturale* riguarda la comprensione di ciò che i LLM stanno realmente ottimizzando internamente: la ricerca di interpretabilità meccanicistica (§433–436) tenta di rispondere a questa domanda decompilando i circuiti computazionali dei transformer in componenti comprensibili.

413.2 · Jailbreaking, Adversarial Prompting e Difese

Il **jailbreaking** è la pratica di costruire prompt che inducono un LLM allineato a produrre output che il suo training dovrebbe impedire. Gli attacchi principali si dividono in categorie per meccanismo. Gli attacchi *di ruolo* chiedono al modello di "impersonare" un sistema senza vincoli (DAN, "Do Anything Now"), sfruttando la capacità del modello di seguire le istruzioni di ruolo. Gli attacchi *di codifica* riformulano la richiesta in base64, ROT13, o linguaggi non ampiamente presenti nel corpus di fine-tuning per sicurezza. Gli attacchi *avversariali ottimizzati* (GCG, Zou et al. 2023) aggiungono automaticamente suffissi di token che massimizzano la probabilità di risposte "compliance" minimizzando la probabilità di rifiuto, tramite ottimizzazione su token discreti con tecniche di token-level gradients. La **robustezza all'avversario nei LLM** è fondamentalmente più difficile che nelle reti di classificazione perché lo spazio di input è discreto e aperto: qualunque difesa basata sul filtraggio del contesto può essere aggirata riformulando il prompt. Le difese più promettenti includono il **constitutional AI training** che addestra esplicitamente il modello a riconoscere e resistere a tentativi di manipolazione, e il **safety-specific reward modeling** che pesa diversamente i falsi negativi (risposta pericolosa non rifiutata) e i falsi positivi (risposta sicura rifiutata) nelle preferenze di allineamento.

413.3 · Interpretabilità Meccanicistica come Strumento di Sicurezza

Un approccio complementare all'allineamento comportamentale è l'**interpretabilità meccanicistica**: comprendere come il modello rappresenta internamente i concetti per rilevare e correggere rappresentazioni pericolose prima che si manifestino nell'output. Tecniche di probing (§435) identificano le direzioni nello spazio delle attivazioni associate a concetti come "inganno", "rifiuto" o "conformità" — direzioni che possono poi essere monitorate durante l'inferenza o modificate tramite **activation steering**. Zou et al. (2023) dimostrano che è possibile estrarre una direzione "disonestà/onestà" nello spazio residual stream dei Transformer e usarla per controllare il comportamento del modello senza alcun fine-tuning. Il **concept bottleneck** e i

metodi di sparse autoencoding (Cunningham et al., 2023) decompongono le attivazioni in feature interpretabili, ciascuna corrispondente a un concetto leggibile: questo permette un audit delle rappresentazioni interne potenzialmente più affidabile delle sole valutazioni comportamentali. La **scalable oversight** (Bowman et al., 2022) affronta il problema dell'allineamento quando i modelli superano le capacità umane di valutazione: tecniche come Debate (modelli avversariali che argomentano mentre un giudice umano valuta) e Recursive Reward Modeling permettono la supervisione umana di task impossibili da valutare direttamente. Il campo dell'allineamento rimane aperto su questioni fondamentali: come specificare utilità umane complesse, come evitare la *reward hacking* su proxy, e come garantire che il comportamento allineato persista quando il modello viene distribuito in contesti diversi da quelli di valutazione.

Score Matching e Modelli Basati sull'Energia

Stimare la funzione di score senza la costante di normalizzazione — le due forme di score matching di Hyvärinen e la connessione con la denoisinig.

414.1 · Lo Score di una Distribuzione e il Problema della Normalizzazione

Un **modello basato sull'energia** (Energy-Based Model, EBM) parametrizza la densità di probabilità come $p_\theta(\mathbf{x}) = e^{-E_\theta(\mathbf{x})}/Z(\theta)$ dove $E_\theta : \mathbb{R}^d \to \mathbb{R}$ è la funzione di energia (tipicamente una rete neurale) e $Z(\theta) = \int e^{-E_\theta(\mathbf{x})}\, d\mathbf{x}$ è la costante di normalizzazione. Il vantaggio degli EBM è la libertà architetturale: qualunque rete neurale positiva può essere usata come funzione di energia. Il problema fondamentale è che $Z(\theta)$ è computazionalmente intrattabile per distribuzioni ad alta dimensione — il calcolo dell'integrale richiede metodi MCMC costosi. La stima della massima verosimiglianza è quindi difficile: $\nabla_\theta \log p_\theta(\mathbf{x}) = -\nabla_\theta E_\theta(\mathbf{x}) - \nabla_\theta \log Z(\theta)$, e il secondo termine richiede il calcolo dell'aspettativa $\nabla_\theta \log Z(\theta) = -\mathbb{E}_{p_\theta}[\nabla_\theta E_\theta(\mathbf{x})]$ rispetto alla distribuzione corrente del modello — un calcolo intrattabile senza campionamento MCMC.

Lo **score di una distribuzione** è il gradiente del logaritmo della densità rispetto all'input: $\mathbf{s}_\theta(\mathbf{x}) = \nabla_\mathbf{x} \log p_\theta(\mathbf{x}) = -\nabla_\mathbf{x} E_\theta(\mathbf{x})$. La costante di normalizzazione scompare prendendo il gradiente rispetto a $\mathbf{x}$ invece che rispetto a θ: $\nabla_\mathbf{x} \log Z(\theta) = 0$ poiché Z non dipende da $\mathbf{x}$. Stimare lo score direttamente è quindi molto più semplice che stimare la densità intera.

414.2 · Score Matching Esplicito e Denoising Score Matching

Hyvärinen (2005) ha mostrato che si può stimare lo score senza calcolare la costante di normalizzazione minimizzando la **Fisher divergence** tra lo score del modello e lo score dei

dati: $\mathcal{J}_{\text{SM}}(\theta) = \frac{1}{2}\mathbb{E}_{p_{\text{data}}}[\|\mathbf{s}_\theta(\mathbf{x}) - \nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})\|^2]$. Poiché lo score dei dati $\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})$ è ignoto, si usa l'integrazione per parti per ottenere un obiettivo dipendente solo dai dati: $\mathcal{J}_{\text{SM}}(\theta) = \mathbb{E}_{p_{\text{data}}}\left[\text{tr}(\nabla_{\mathbf{x}} \mathbf{s}_\theta(\mathbf{x})) + \frac{1}{2}\|\mathbf{s}_\theta(\mathbf{x})\|^2\right] + \text{const}$ dove $\text{tr}(\nabla_{\mathbf{x}} \mathbf{s}_\theta)$ è la traccia della matrice jacobiana dello score — computazionalmente costosa in alta dimensione. Il **Denoising Score Matching (DSM)** (Vincent 2011, Alain e Bengio 2014) offre un'alternativa pratica: si aggiunge rumore gaussiano $\tilde{\mathbf{x}} = \mathbf{x} + \boldsymbol{\epsilon}$ con $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2 I)$ ai dati, e si addestra il modello a stimare lo score della distribuzione rumorosa $p_\sigma(\tilde{\mathbf{x}}) = \int p_{\text{data}}(\mathbf{x}) \mathcal{N}(\tilde{\mathbf{x}}; \mathbf{x}, \sigma^2 I)\, d\mathbf{x}$. Lo score di questa distribuzione rumorosa ha la forma $\nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}}) = -\mathbb{E}[\boldsymbol{\epsilon}|\tilde{\mathbf{x}}]/\sigma^2 = (\mathbf{x} - \tilde{\mathbf{x}})/\sigma^2$ — proporzionale alla direzione di denoise. L'obiettivo di training diventa quindi la predizione del rumore aggiunto: $\mathcal{J}_{\text{DSM}}(\theta) = \mathbb{E}_{\mathbf{x},\boldsymbol{\epsilon}}[\|\mathbf{s}_\theta(\tilde{\mathbf{x}}, \sigma) + \boldsymbol{\epsilon}/\sigma\|^2]$ — un semplice obiettivo di regressione sul rumore, che elimina completamente il calcolo della traccia jacobiana.

DDPM: Derivazione Matematica Completa

Il processo di diffusione forward e reverse — la variational lower bound, la parametrizzazione del rumore e il collegamento con il score matching.

415.1 · Il Forward Process: Aggiungere Rumore in Modo Controllato

I **Denoising Diffusion Probabilistic Models (DDPM)** (Ho, Jain e Abbeel 2020) strutturano la generazione come l'inversione di un processo di corruzione progressiva dei dati. Il **forward process** aggiunge rumore gaussiano in T passi con un programma di varianza $\beta_1, \ldots, \beta_T \in (0, 1)$:

$$q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t;\ \sqrt{1 - \beta_t}\, \mathbf{x}_{t-1},\ \beta_t I)$$

La proprietà chiave di questo processo è che si può campionare $\mathbf{x}_t$ direttamente da $\mathbf{x}_0$ senza iterare tutti i passi intermedi. Definendo $\alpha_t = 1 - \beta_t$ e $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$, si ottiene: $q(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t;\ \sqrt{\bar{\alpha}_t}\, \mathbf{x}_0,\ (1 - \bar{\alpha}_t) I)$ oppure equivalentemente $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\, \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\, \boldsymbol{\epsilon}$ con $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, I)$. Per T abbastanza grande e β_t scelti opportunamente (ad esempio con un cosine schedule), si ha $\bar{\alpha}_T \approx 0$ e $\mathbf{x}_T \approx \mathcal{N}(\mathbf{0}, I)$ — puro rumore gaussiano, indipendente da $\mathbf{x}_0$.

415.2 · Il Reverse Process e la ELBO

Il **reverse process** è la distribuzione markoviana che inverte il forward process: $p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1};\, \boldsymbol{\mu}_\theta(\mathbf{x}_t, t),\, \Sigma_\theta(\mathbf{x}_t, t))$. Per addestrare θ, si massimizza il log-likelihood $\log p_\theta(\mathbf{x}_0) = \log \int p_\theta(\mathbf{x}_{0:T})\, d\mathbf{x}_{1:T}$ — intrattabile direttamente. L'ELBO variazionale con q come distribuzione approssimativa forward scompone la loss in tre termini:

$$-\log p_\theta(\mathbf{x}_0) \leq \underbrace{\mathbb{E}_q[\mathrm{KL}(q(\mathbf{x}_T|\mathbf{x}_0)\|p_\theta(\mathbf{x}_T))]}_{\text{prior matching}} + \underbrace{\sum_{t>1} \mathbb{E}_q[\mathrm{KL}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)\|p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))]}_{\text{denoising}} - \underbrace{\mathbb{E}_q[\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)]}_{\text{ricostruzione}}$$

Il termine dominante è quello di denoising. La distribuzione $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ è trattabile per il forward process gaussiano — ha forma gaussiana con media $\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t$. Sostituendo la relazione $\mathbf{x}_0 = (\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon})/\sqrt{\bar{\alpha}_t}$ e parametrizzando il modello come predittore del rumore $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$, la loss di denoising si semplifica a una loss MSE sul rumore previsto:

$$\mathcal{L}_{\text{simple}}(\theta) = \mathbb{E}_{t, \mathbf{x}_0, \boldsymbol{\epsilon}}\left[\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon},\, t)\|^2\right]$$

Questa loss, chiamata la *simplified loss* di DDPM, è notevolmente semplice ed è equivalente al Denoising Score Matching con livelli di rumore multipli: il modello apprende a predire il rumore aggiunto, e la direzione negativa del rumore predetto è proporzionale allo score della distribuzione rumorosa, $\boldsymbol{\epsilon}_\theta \approx -\sqrt{1-\bar{\alpha}_t}\,\nabla_{\mathbf{x}_t} \log p_{\bar{\alpha}_t}(\mathbf{x}_t)$.

DDIM e Accelerazione del Campionamento

Generazione deterministica e salto di passi — la derivazione del DDIM, il tradeoff qualità-velocità e i campionatori ODE moderni.

416.1 · La Formulazione ODE del Campionamento Diffusivo

Il DDPM originale richiede $T = 1000$ passi di denoising per generare un'immagine — troppo lento per applicazioni pratiche. Il **DDIM (Denoising Diffusion Implicit Models)** (Song, Meng e Ermon 2021) mostra che esiste una famiglia di processi non markoviani con la stessa

distribuzione marginale $q(\mathbf{x}_t|\mathbf{x}_0)$ del DDPM ma che può essere invertita in modo deterministic con pochi passi. Il processo DDIM definisce un reverse step:

$$\mathbf{x}_{t-1} = \sqrt{\bar{\alpha}_{t-1}} \underbrace{\left(\frac{\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)}{\sqrt{\bar{\alpha}_t}} \right)}_{\text{predizione } \hat{\mathbf{x}}_0} + \sqrt{1-\bar{\alpha}_{t-1}-\sigma_t^2}\,\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) + \sigma_t \boldsymbol{\epsilon}$$

Con $\sigma_t = 0$ per tutti i t, il processo diventa completamente deterministico: la traiettoria di generazione è una funzione deterministica del rumore iniziale $\mathbf{x}_T$, e si può interpolare smoothly tra due immagini interpolando i loro $\mathbf{x}_T$. Con $\sigma_t > 0$ si recupera il comportamento stocastico del DDPM. La formulazione DDIM corrisponde alla discretizzazione di un'ODE di probabilità di flusso (probability flow ODE): $\frac{d\mathbf{x}}{dt} = -\frac{1}{2}\dot{\sigma}(t)\sigma(t)\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$ che genera la stessa distribuzione marginale della SDE stocastica di diffusione ma in modo deterministic, ammettendo solutori ODE a alto ordine (Runge-Kutta, Adams-Bashforth) che convergono con molti meno passi. Solutori come DEIS, DPM-Solver++ e UniPC ottengono qualità paragonabile a DDPM con 10–20 passi invece di 1000, una riduzione di 50–100x nel costo computazionale dell'inferenza.

416.2 · Campionatori Avanzati e Distillazione a Passo Singolo

La letteratura sui **solutori ODE per diffusion model** si è sviluppata rapidamente. DPM-Solver (Lu et al., 2022) sfrutta la struttura semi-lineare dell'ODE di flusso: la soluzione formale si decompone in una parte lineare integrabile in forma chiusa e una parte non lineare approssimata con Taylor. DPM-Solver++ parametrizza la predizione come $\hat{\mathbf{x}}_0$ invece del rumore $\boldsymbol{\epsilon}$, ottenendo convergenza più stabile a pochi passi. UniPC (Zhao et al., 2023) unifica diversi solutori in un framework predictor-corrector generalizzato: il predictor stima $\mathbf{x}_{t-1}$ con un metodo multi-step, il corrector raffina sfruttando la valutazione della rete già disponibile senza costo aggiuntivo. Con 5–10 passi UniPC raggiunge la qualità del DDPM con 1000 passi. La **consistency distillation** (Song et al., 2023) porta il costo a un singolo passo imponendo $f_\theta(\mathbf{x}_t, t) = f_\theta(\mathbf{x}_{t'}, t')$ per ogni coppia (t, t') sulla stessa traiettoria ODE: il risultato è un generatore one-step con qualità comparabile ai campionatori multi-step, al costo di una singola forward pass.

Il **tradeoff qualità-velocità** nei diffusion model moderni è governato da tre variabili: NFE (number of function evaluations), tipo di campionatore e guidance scale w. Ridurre NFE introduce errori di discretizzazione; aumentare w migliora l'aderenza al condizionamento ma riduce la diversità. I **distilled one-step models** come ADD (Adversarial Diffusion Distillation, Sauer et al., 2023) combinano distillazione e training avversariale: il generatore one-step

viene addestrato a produrre immagini indistinguibili dal modello teacher multi-step tramite un discriminatore, raggiungendo qualità paragonabile a Stable Diffusion XL a 20–40 passi usando un singolo forward pass. La generazione su device mobili e edge hardware beneficia direttamente di questi progressi: modelli come LCM-LoRA combinano Latent Consistency Model con LoRA per generare immagini 512×512 in <1 secondo su hardware consumer.

Score-Based SDE di Song: il Framework Unificato

La diffusione come SDE — VE, VP e sub-VP, la probability flow ODE e il campionamento predictor-corrector.

417.1 · Il Framework SDE per la Diffusione

Song, Sohl-Dickstein, Kingma, Kumar, Ermon e Poole (2021, NeurIPS Best Paper) hanno unificato i modelli DDPM, NCSN (Noise Conditional Score Network) e i loro predecessori in un framework continuo basato sulle equazioni differenziali stocastiche. Il forward process è una SDE di Itô generale: $d\mathbf{x} = \mathbf{f}(\mathbf{x}, t)\,dt + g(t)\,d\mathbf{W}$ dove $\mathbf{f}$ è il drift, g la diffusione, e $\mathbf{W}$ il moto browniano standard in $\mathbb{R}^d$. Il **reverse SDE di Anderson (1982)** inverte il tempo di questa SDE: $d\mathbf{x} = [\mathbf{f}(\mathbf{x}, t) - g(t)^2 \nabla_{\mathbf{x}} \log p_t(\mathbf{x})]\,dt + g(t)\,d\bar{\mathbf{W}}$ dove $d\bar{\mathbf{W}}$ è il moto browniano standard nel tempo inverso e $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$ è lo score della distribuzione marginale al tempo t. Se si addestra una rete neurale $\mathbf{s}_\theta(\mathbf{x}, t) \approx \nabla_{\mathbf{x}} \log p_t(\mathbf{x})$, si può campionare dalla distribuzione dei dati simulando la reverse SDE da $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, I)$ fino a $\mathbf{x}_0$. Le due famiglie principali sono la **Variance Exploding (VE) SDE** ($\mathbf{f} = \mathbf{0}$, $g(t) = \sigma_{\min}(\sigma_{\max}/\sigma_{\min})^t \sqrt{2\log(\sigma_{\max}/\sigma_{\min})}$, corrispondente a NCSN) e la **Variance Preserving (VP) SDE** ($\mathbf{f}(\mathbf{x}, t) = -\frac{1}{2}\beta(t)\mathbf{x}$, corrispondente a DDPM). In entrambi i casi, l'addestramento dello score network si riduce allo stesso obiettivo DSM pesato su livelli di rumore diversi: $\mathbb{E}_{t,\mathbf{x}_0,\boldsymbol{\epsilon}}[\lambda(t)\|\mathbf{s}_\theta(\mathbf{x}_t, t) - \nabla_{\mathbf{x}_t} \log q(\mathbf{x}_t|\mathbf{x}_0)\|^2]$ dove $\lambda(t)$ è una funzione di peso che bilancia i diversi livelli di rumore.

417.2 · VP, VE e sub-VP: una Famiglia di Processi Diffusivi

Song et al. (2021) identificano tre istanze principali del framework SDE. Il processo **Variance Preserving (VP)** — corrispondente al DDPM — usa $d\mathbf{x} = -\frac{1}{2}\beta(t)\mathbf{x}\,dt + \sqrt{\beta(t)}\,d\mathbf{w}$: la varianza rimane approssimativamente unitaria grazie al drift lineare. Il processo **Variance Exploding (VE)** — corrispondente a NCSN — usa $d\mathbf{x} = \sigma(t)\,d\mathbf{w}$ senza drift: la varianza esplode da $\sigma_{\min}^2$ a $\sigma_{\max}^2$. Il processo **sub-VP** garantisce che la covarianza rimanga strettamente inferiore a quella del VP, migliorando la stima dello score ad alti t. Il teorema di Anderson (1982) garantisce che

per qualunque SDE in avanti esiste un processo reverso con la stessa distribuzione marginale governato da $d\mathbf{x} = [\mathbf{f} - g^2\nabla_{\mathbf{x}} \log p_t]\, dt + g\, d\bar{\mathbf{w}}$. La rete neurale approssima $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$ per tutti i valori di t simultaneamente tramite score matching pesato, rendendo il framework applicabile a qualsiasi scelta di SDE in avanti.

Il **predictor-corrector sampling** combina solutori della reverse SDE (predictor) con passi di Langevin dynamics (corrector): dopo ogni passo di reverse SDE, si eseguono M passi di annealed Langevin MCMC per correggere gli errori di discretizzazione. La scelta tra SDE e ODE impatta la diversità dei campioni: la SDE introduce rumore che esplora l'intera distribuzione, l'ODE deterministica converge sempre allo stesso output dato lo stesso $\mathbf{x}_T$. Il framework SDE unificato trova applicazioni in video generation (Video Diffusion), audio synthesis (DiffWave), protein structure generation (RFDiffusion) e molecular conformer generation — adattando la SDE alla struttura geometrica specifica di ogni dominio.

Latent Diffusion e Stable Diffusion

Diffusione nello spazio latente — il VQ-regularized autoencoder, il cross-attention per il condizionamento e la scala dei modelli text-to-image.

418.1 · Diffusione nello Spazio Latente

Applicare il processo di diffusione direttamente allo spazio dei pixel è computazionalmente costoso: un'immagine 512×512×3 ha oltre 786.000 dimensioni, e il calcolo di $T = 1000$ passi di U-Net su questa dimensione richiede risorse enormi. I **Latent Diffusion Models (LDM)** (Rombach, Blattmann, Lorenz, Esser e Ommer 2022, la base di Stable Diffusion) risolvono il problema separando la compressione percettiva dalla generazione: un autoencoder (tipicamente VQ-VAE o KL-regularized VAE) comprime le immagini in un spazio latente $\mathbf{z} = \mathcal{E}(\mathbf{x}) \in \mathbb{R}^{h \times w \times c}$ con $h, w \ll H, W$ e c piccolo (tipicamente 4 canali), e il processo di diffusione opera su $\mathbf{z}$ invece che su $\mathbf{x}$. Il decoder $\mathcal{D}(\mathbf{z})$ ricostruisce l'immagine dallo spazio latente. Il vantaggio della diffusione latente è la riduzione dimensionale di due ordini di grandezza: invece di diffondere in $H \times W \times 3 \approx 786.432$ dimensioni, si diffonde in $h \times w \times c \approx 4096$ – una riduzione del 99%. Questo rende praticabile addestrare modelli di diffusione di alta qualità su hardware consumer.

Il **condizionamento al testo** in Stable Diffusion usa un meccanismo di **cross-attention** nel U-Net denoiser: ogni residual block del U-Net include un layer di cross-attention che prende come queries le feature spaziali dell'immagine latente e come keys/values le codifiche del testo prodotte da un encoder linguistico (CLIP text encoder o OpenCLIP). La cross-attention formula

è $\text{CrossAttn}(\mathbf{z}, \mathbf{c}) = \text{softmax}\left(\frac{\mathbf{z}W_Q(\mathbf{c}W_K)^T}{\sqrt{d}}\right)\mathbf{c}W_V$ dove $\mathbf{z}$ sono le feature spaziali e $\mathbf{c}$ è la codifica del testo. Questo meccanismo permette al denoiser di "guidare" il processo di denoise tenendo conto del prompt testuale ad ogni passo — producendo immagini semanticamente allineate al testo senza richiedere coppie di addestramento immagine-testo perfettamente allineate grazie al **classifier-free guidance (CFG)**: il modello viene addestrato sia con che senza condizionamento (con probabilità 10–20% il condizionamento viene azzerato), e all'inferenza si estrapolano le predizioni condizionate verso quelle non condizionate: $\tilde{\boldsymbol{\epsilon}}_\theta = \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, \emptyset) + w \cdot (\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, \mathbf{c}) - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, \emptyset))$ dove $w > 1$ è il guidance scale.

418.2 · Architetture di Condizionamento: Cross-Attention e DiT

Il condizionamento testuale in Stable Diffusion avviene attraverso **cross-attention** nei layer della U-Net: le chiavi e i valori provengono dall'embedding testuale prodotto da CLIP o T5, mentre le query provengono dalle feature immagine. Questo permette di controllare la generazione a diversi livelli di astrazione: layer profondi governano la composizione semantica globale, layer superficiali i dettagli spaziali locali. L'architettura **DiT (Diffusion Transformer)** (Peebles e Xie, 2023) sostituisce la U-Net con un Vision Transformer puro: le patch immagine vengono trattate come token in sequenza, il condizionamento su t e c viene iniettato via adaptive layer norm (adaLN), e l'architettura scala più prevedibilmente con le leggi di scala rispetto alla U-Net convoluzionale. Stable Diffusion 3 e FLUX adottano architetture MM-DiT con flussi paralleli per testo e immagine che si fondono tramite full attention, migliorando la fedeltà al prompt su composizioni complesse. Il **ControlNet** (Zhang et al., 2023) aggiunge condizionamento spaziale preciso (profondità, pose, bordi Canny) copiando i layer encoder della U-Net in una rete parallela addestrata separatamente, preservando le capacità generative del modello base.

Consistency Models e Distillazione dei Modelli di Diffusione

Generazione in un solo passo — la consistency function, il consistency training e la distillazione progressiva.

419.1 · Il Problema dei Mille Passi e la Consistency Function

Nonostante i solutori ODE accelerati abbiano ridotto i passi da 1000 a 10–20, la generazione diffusiva rimane più lenta di un singolo forward pass di GAN o VAE. I **Consistency Models** (Song, Dhariwal, Chen e Sutskever 2023, OpenAI) affrontano questo problema imponendo

direttamente la coerenza della mappa da qualunque punto della traiettoria ODE all'origine: si addestra una funzione $f_\theta(\mathbf{x}_t, t)$ con la proprietà di *self-consistency* $f_\theta(\mathbf{x}_t, t) = f_\theta(\mathbf{x}_{t'}, t')$ per qualunque coppia $(\mathbf{x}_t, \mathbf{x}_{t'})$ sulla stessa traiettoria ODE. Se questa proprietà è soddisfatta, si può generare un campione in un singolo passo: $\mathbf{x}_0 = f_\theta(\mathbf{x}_T, T)$ da qualunque $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, I)$. La self-consistency si impone con una loss che penalizza la discrepanza tra stime consecutive lungo la traiettoria discretizzata: $\mathcal{L}_{\text{CT}}(\theta, \theta^-) = \mathbb{E}_{t,\mathbf{x}_0,\boldsymbol{\epsilon}}\Big[d(f_\theta(\mathbf{x}_{t_{n+1}}, t_{n+1}),\, f_{\theta^-}(\hat{\mathbf{x}}^\phi_{t_n}, t_n))\Big]$ dove $\hat{\mathbf{x}}^\phi_{t_n}$ è un passo dell'ODE numerica da $\mathbf{x}_{t_{n+1}}$, θ^- è un EMA di θ (come nella rete target del DQN), e d è una distanza robusta. I Consistency Models possono essere addestrati sia da zero (Consistency Training) sia come distillazione di un modello diffusivo pre-addestrato (Consistency Distillation), e producono qualità comparabile a DDIM con 10 passi in un singolo step di inferenza — una riduzione del costo di un ordine di grandezza ulteriore rispetto ai solutori ODE.

419.2 · Flow Matching: Traiettorie Rette come Alternativa alla Diffusione

Il **Flow Matching** (Lipman et al., 2022) apprende un campo vettoriale $v_\theta(\mathbf{x}, t)$ che trasporta la distribuzione del rumore verso quella dei dati lungo traiettorie interpolanti. L'obiettivo di training è $\mathcal{L} = \mathbb{E}_{t,\mathbf{x}_0,\mathbf{x}_1}[\|v_\theta(\mathbf{x}_t, t) - (\mathbf{x}_1 - \mathbf{x}_0)\|^2]$ dove $\mathbf{x}_t = t\mathbf{x}_1 + (1-t)\mathbf{x}_0$ interpola linearmente. Rispetto alla diffusione, il flow matching produce traiettorie più dirette — il che si traduce in convergenza con meno passi ODE a parità di qualità, e una loss di training più stabile. **Rectified Flow** aggiunge rettilineazione iterativa: si campionano coppie $(\mathbf{x}_0, \mathbf{x}_1)$ usando il modello corrente e si ri-addestra il modello a seguire le traiettorie rette risultanti, riducendo iterativamente la curvatura del flusso e abilitando la generazione one-step di alta qualità. Il flow matching è ora alla base di modelli come Stable Diffusion 3 e FLUX, che usano traiettorie rettificate per generazione di immagini e video ad alta fedeltà.

La valutazione dei generative model usa metriche standardizzate. Il **FID (Fréchet Inception Distance)** misura la distanza di Fréchet tra le distribuzioni di feature InceptionV3 di immagini reali e generate: FID basso indica alta fedeltà e diversità. Il **CLIP Score** misura l'allineamento testo-immagine. Su ImageNet 256×256 i migliori modelli di diffusione (DiT-XL/2, SDXL) raggiungono FID < 2, paragonabile ai migliori GAN ma con migliore coverage della distribuzione e senza mode collapse. Il **Perceptual Path Length (PPL)**, mutuato dai GAN, misura la smoothness dello spazio latente valutando quanto cambiano le immagini al variare di piccoli step nel latent space — una misura indiretta della struttura e della disentanglement della rappresentazione latente.

Message Passing e il Framework MPNN

L'astrazione comune a GCN, GraphSAGE, GAT e MPNN — la funzione di aggregazione, di aggiornamento e la lettura del grafo.

420.1 · Il Framework Message Passing

Le **Graph Neural Networks (GNN)** operano su grafi $G = (V, E)$ dove ogni nodo $v \in V$ ha una feature iniziale $\mathbf{h}_v^{(0)} \in \mathbb{R}^d$ e ogni arco $(u, v) \in E$ può avere un attributo $\mathbf{e}_{uv}$. Il framework **Message Passing Neural Network (MPNN)** (Gilmer et al. 2017) descrive l'aggiornamento delle rappresentazioni nodali tramite tre operazioni fondamentali che si ripetono per K iterazioni (layer). Il **passo di messaggio** calcola i messaggi che ogni nodo invia ai propri vicini: $\mathbf{m}_{uv}^{(k)} = \phi^{(k)}\Big(\mathbf{h}_v^{(k-1)}, \mathbf{h}_u^{(k-1)}, \mathbf{e}_{uv}\Big)$ dove $\phi^{(k)}$ è una rete neurale (tipicamente MLP). Il **passo di aggregazione** raccoglie i messaggi dai vicini: $\mathbf{a}_v^{(k)} = \bigoplus_{u \in \mathcal{N}(v)} \mathbf{m}_{uv}^{(k)}$ dove $\bigoplus$ è una funzione di aggregazione permutation-invariant (somma, media, max, o attenzione). Il **passo di aggiornamento** combina il messaggio aggregato con la rappresentazione corrente del nodo: $\mathbf{h}_v^{(k)} = \psi^{(k)}\Big(\mathbf{h}_v^{(k-1)}, \mathbf{a}_v^{(k)}\Big)$ dove $\psi^{(k)}$ è un'altra rete neurale o una GRU. Dopo K layer, una funzione di **readout** permutation-invariant produce una rappresentazione del grafo intero: $\mathbf{h}_G = R\Big(\Big\{\mathbf{h}_v^{(K)}\Big\}_{v \in V}\Big)$ dove R può essere una somma, una media, o meccanismi più sofisticati come la Sum-DeepSets o la Graph Multi-Set Transformer. GCN (Kipf e Welling 2017), GraphSAGE (Hamilton et al. 2017), e GAT (Veličković et al. 2018) sono casi speciali di questo framework che differiscono principalmente nella scelta dell'aggregatore $\bigoplus$ e nella parametrizzazione di ϕ e ψ.

420.2 · Aggregatori e Pooling Gerarchico per Grafi

La scelta dell'aggregatore $\bigoplus$ determina le proprietà di invarianza e l'espressività del modello. La **somma** è l'aggregatore più espressivo per distinguere strutture multiset: se due nodi hanno quartierati diversi, la somma dei messaggi produce valori diversi anche con feature identiche, a differenza della media che confonde quartierati di cardinalità differente con feature scalate proporzionalmente. La **max-pooling** è invariante alla molteplicità e cattura la feature più prominente del quartierato, utile quando interessa la presenza di un pattern anziché la sua frequenza. Il **LSTM aggregator** (Hamilton et al., 2017) applica un LSTM ai messaggi dei vicini in ordine casuale — non è strettamente invariante alla permutazione ma in pratica mostra buona stabilità su grafi densi. Per la classificazione di grafi interi, il **pooling gerarchico** costruisce una coarsened version del grafo a ogni livello: DiffPool (Ying et al., 2018) apprende un'assegnazione differenziabile dei nodi a k cluster usando una matrice di assegnazione softmax $S \in \mathbb{R}^{nimesk}$, producendo un grafo con k super-nodi per il livello successivo. Il costo $O(n^2)$ di

DiffPool viene mitigato da approcci sparsi come MinCutPool e ASAPool che operano localmente sul grafo di vicinanza.

La scelta tra **pooling globale** e **pooling gerarchico** dipende dal task: per la classificazione di grafi piccoli (molecole, reti sociali) la semplice somma o media delle rappresentazioni nodali finali è spesso sufficiente e computazionalmente vantaggiosa. Per grafi grandi con struttura gerarchica (scene graph, proteine) il pooling gerarchico cattura strutture multi-scala che la pooling piatta non può rappresentare. Il **Set2Set** (Vinyals et al., 2015) usa un LSTM con attenzione sull'insieme dei nodi per produrre una rappresentazione globale order-invariant che tiene conto dell'interazione tra nodi — più espressivo della semplice somma ma con complessità $O(n \cdot d)$ per ogni passo dell'LSTM.

Espressività delle GNN: il Test di Weisfeiler-Lehman e GIN

Quali grafi una GNN riesce a distinguere — il collegamento tra MP e WL, la GNN massimalmente espressiva e gli Higher-Order GNN.

421.1 · Il Limite Espressivo del Message Passing Standard

Xu, Hu, Leskovec e Jegelka (2019, ICLR Best Paper) hanno dimostrato un risultato fondamentale sull'espressività delle GNN: qualunque GNN con message passing standard non può distinguere grafi che il **test di Weisfeiler-Lehman (1-WL)** non riesce a distinguere, e la GNN più espressiva raggiungibile con message passing standard è quella che simula esattamente il test 1-WL. Il test 1-WL itera: per ogni nodo, si raccoglie il multi-insieme delle label dei vicini e si calcola una nuova label tramite hashing; si ripete fino a stabilizzazione. Due grafi sono considerati non isomorfi se in qualunque iterazione la distribuzione di label di uno differisce dall'altro. La prova del bound superiore è per simulazione: ogni GNN con aggregazione iniettiva simula un passo del 1-WL. La prova del bound inferiore mostra che grafi che 1-WL non distingue (ad esempio, due cicli di lunghezza 6 con struttura simmetrica) non possono essere distinti da nessuna GNN a message passing standard, indipendentemente dalla profondità o dall'espressività delle funzioni ϕ e ψ. La **Graph Isomorphism Network (GIN)** è costruita per raggiungere questo massimo espressivo: usa la somma come aggregatore (invece della media, che comprime informazione) e un'architettura MLP iniettiva: $\mathbf{h}_v^{(k)} = \mathrm{MLP}^{(k)}\Big((1+\epsilon^{(k)})\mathbf{h}_v^{(k-1)} + \sum_{u\in\mathcal{N}(v)} \mathbf{h}_u^{(k-1)}\Big)$ dove ϵ è un parametro apprendibile. La somma pesata con $(1+\epsilon)$ garantisce che il nodo distingua il proprio contributo da quello dei vicini — proprietà necessaria per l'iniettività dell'aggregazione.

421.2 · Higher-Order GNN: Oltre il Limite WL

Per superare il limite 1-WL, le **Higher-Order GNN** usano il test k-WL che opera su tuple di k nodi invece di singoli nodi. Il k-WL distingue più strutture del $(k-1)$-WL per ogni k, al costo di complessità $O(n^k)$ in numero di parametri e $O(n^k)$ in complessità computazionale. Le versioni pratiche usano $k = 2$ o $k = 3$ con ottimizzazioni che riducono il costo computazionale tramite sparsità. Un'alternativa più pratica è l'aggiunta di informazioni strutturali non locali: contare i cicli di lunghezza 3, 4, 5 (triangoli, quadrilateri, cicli più lunghi), incorporare la posizione spettrale dei nodi (gli autovettori della matrice Laplaciana), o usare la distanza topologica tra coppie di nodi come feature aggiuntiva. Le **Random Walk GNN** codificano la struttura locale tramite la distribuzione della passeggiate casuali di diversa lunghezza che partono da ogni nodo — una feature che cattura informazioni sul quartierato locale senza calcolare esplicitamente il k-WL.

421.3 · GNN per Strutture Irregolari: Eterogeneità e Ipergrafi

I grafi del mondo reale raramente sono omogenei: i grafi di conoscenza (Knowledge Graphs) hanno nodi e archi di tipo eterogeneo, i grafi di citazione hanno distinzioni tra paper e autore, i grafi biologici distinguono geni, proteine e metaboliti. Le **Relational Graph Convolutional Networks (R-GCN)** (Schlichtkrull et al., 2018) parametrizzano una matrice di trasformazione separata per ogni tipo di relazione: $h_v^{(l+1)} = \sigma\left(\sum_{r\in\mathcal{R}} \sum_{u\in\mathcal{N}_r(v)} \frac{1}{c_{v,r}} W_r^{(l)} h_u^{(l)}\right)$ dove $\mathcal{N}_r(v)$ sono i vicini di v per la relazione r. Il numero di parametri cresce linearmente con $|\mathcal{R}|$, mitigato da tecniche di decomposizione: basis decomposition esprime $W_r = \sum_b a_{rb} V_b$ con $B \ll |\mathcal{R}|$ matrici di base condivise. Gli **ipergrafi** — dove ogni iper-arco può connettere più di due nodi — richiedono message passing generalizzato che operi attraverso iper-archi: HNN e HNHN definiscono messaggi che aggregano tutti i nodi in un iper-arco prima di distribuirli nuovamente, catturando relazioni d'ordine superiore inaccessibili ai grafi ordinari.

Graph Attention Networks: Aggregazione Selettiva tramite Attenzione

Pesare i vicini in modo differenziabile — la funzione di attenzione di GAT, la multi-head attention sui grafi e la connessione con i transformer.

422.1 · La Meccanica dell'Attenzione nei Grafi

Le **Graph Attention Networks (GAT)** (Veličković, Cucurull, Casanova, Romero, Liò e Bengio 2018) sostituiscono l'aggregazione uniforme (media o somma) con un'aggregazione pesata

dove i pesi dipendono dalla coppia di nodi. Il coefficiente di attenzione tra il nodo v e il vicino u è calcolato come:

$$\alpha_{vu} = \frac{\exp(\text{LeakyReLU}(\mathbf{a}^T[\mathbf{W}\mathbf{h}_v \| \mathbf{W}\mathbf{h}_u]))}{\sum_{w \in \mathcal{N}(v)\cup\{v\}} \exp(\text{LeakyReLU}(\mathbf{a}^T[\mathbf{W}\mathbf{h}_v \| \mathbf{W}\mathbf{h}_w]))}$$

dove $\mathbf{W}$ è una matrice di proiezione condivisa e $\mathbf{a}$ è un vettore di attenzione apprendibile. L'aggiornamento del nodo è $\mathbf{h}_v^{(k)} = \sigma\Big(\sum_{u \in \mathcal{N}(v)\cup\{v\}} \alpha_{vu} \mathbf{W}\mathbf{h}_u\Big)$. La **multi-head attention** esegue H meccanismi di attenzione in parallelo e concatena (o media) i risultati: $\mathbf{h}_v^{(k)} = \|_{h=1}^{H} \sigma\big(\sum_u \alpha_{vu}^h \mathbf{W}^h \mathbf{h}_u\big)$. Il vantaggio fondamentale di GAT rispetto a GCN è la capacità di assegnare importanze diverse ai vicini in base al contenuto delle loro rappresentazioni — particolarmente utile per grafi eterogenei dove i vicini hanno ruoli diversi (ad esempio, in una rete di citazioni, un articolo citato da molti articoli rilevanti dovrebbe pesare di più di uno citato incidentalmente). La **GAT v2** (Brody, Alon e Yahav 2022) corregge un difetto della GAT originale: il coefficiente di attenzione originale è lineare nelle rappresentazioni dei nodi e quindi esprime solo attenzione *statica* (dipende solo da $\mathbf{h}_v$ e $\mathbf{h}_u$ separatamente ma non dalla loro interazione); GAT v2 usa $\mathbf{a}^T \text{LeakyReLU}(\mathbf{W}_l \mathbf{h}_v + \mathbf{W}_r \mathbf{h}_u)$ che è veramente dinamico.

422.2 · Transformers su Grafi e Graph-Level Tasks

L'attenzione dei Transformer è concettualmente compatibile con il message passing sui grafi: entrambi computano una combinazione pesata di messaggi da nodi vicini, ma l'attenzione del Transformer opera su tutti i nodi simultaneamente (attenzione globale) mentre il GAT opera solo sui vicini diretti (attenzione locale). Il **Graph Transformer** unifica i due approcci: l'attenzione locale mantiene la localizzazione strutturale mentre l'attenzione globale (con complessità $O(n^2)$) cattura dipendenze a lungo raggio inaccessibili al message passing k-layer. Graphormer (Ying et al., 2021) introduce bias di posizionamento basati su distanza strutturale: al punteggio di attenzione si aggiunge un termine $b_{\phi(v_i,v_j)}$ dove $\phi(v_i, v_j)$ è la distanza sul cammino più breve tra i nodi i e j, codificando la struttura topologica nel meccanismo di attenzione. Per scalare a grafi con milioni di nodi, **Expander Graph Propagation** e metodi di campionamento come GraphSAINT (Zeng et al., 2020) suddividono il grafo in mini-batch mantenendo la correttezza statistica dell'aggiornamento dei nodi tramite normalizzazione appropriata degli archi campionati.

L'**over-smoothing** nei GNN profondi — le rappresentazioni nodali convergono a un vettore costante dopo molti layer di message passing — è analogo alla diffusione del calore sul grafo: con L layer di GCN, la rappresentazione di ogni nodo incorpora informazioni dal suo L-hop

neighborhood, e per grafi con diametro piccolo L grande produce feature indistinguibili. La regolarizzazione tramite **DropEdge** (rimozione casuale di archi durante il training), **residual connections** e **jumping knowledge networks** (Xu et al., 2018, che concatenano le rappresentazioni di tutti i layer) mitigano l'over-smoothing permettendo GNN più profondi con campo recettivo maggiore senza perdita di specificità nodale.

GNN per le Molecole e l'Equivarianza E(3)

Grafi tridimensionali, invarianza alle isometrie e le architetture SchNet, DimeNet, PaiNN e EGNN per la chimica computazionale.

423.1 · La Struttura delle Molecole come Grafo Tridimensionale

Una molecola è naturalmente un grafo i cui nodi sono atomi (con feature come tipo atomico, numero di valenza) e i cui archi rappresentano legami chimici o distanze spaziali. A differenza dei grafi di citazione o dei social network, le molecole hanno una struttura geometrica tridimensionale: le coordinate $\mathbf{r}_i \in \mathbb{R}^3$ di ogni atomo sono parte essenziale della struttura e determinano proprietà chimiche fondamentali. Il vincolo critico è l'**equivarianza al gruppo euclideo E(3)** (o SE(3) per le sole rotazioni e traslazioni, senza riflessioni): la predizione di scalari chimici (energia, attività biologica) deve essere invariante a traslazioni, rotazioni e riflessioni della molecola — la stessa molecola orientata diversamente ha le stesse proprietà; la predizione di vettori (forze atomiche, dipolo) deve essere equivariante. I metodi che ignorano la geometria 3D e usano solo la topologia del legame (GNN standard) perdono informazioni chirali e conformazionali che sono essenziali per la biologia e la chimica dei materiali. SchNet (Schütt et al. 2017) usa distanze interatomiche come feature degli archi; DimeNet (Klicpera et al. 2020) aggiunge angoli di bond; SphereNet aggiunge anche angoli torsionali (diedri). **EGNN** (Satorras, Hoogeboom e Welling 2021) è equivariante a E(3) per costruzione: gli aggiornamenti delle coordinate usano solo differenze vettoriali relativa $(\mathbf{r}_i - \mathbf{r}_j)$, garantendo l'equivarianza senza calcolo delle rappresentazioni irriducibili. Le architetture basate sulle rappresentazioni irriducibili di SO(3) come e3nn, NequIP e MACE raggiungono la massima espressività equivariante per la modellazione molecolare, con applicazioni a potenziali interatomici di machine learning per simulazioni di dinamica molecolare a scala ab initio.

423.2 · Invarianza vs. Equivarianza: Coordinate Locali e SE(3)-GNN

La distinzione tra **invarianza** ed **equivarianza** è fondamentale per le applicazioni molecolari. Un modello invariante produce lo stesso output per molecole ruotate/traslate — necessario per classificazione e regressione di energia. Un modello equivariante trasforma l'output in modo consistente con la trasformazione applicata — necessario per predire forze, momenti dipolari e altre quantità vettoriali o tensoriali. SchNet e DimeNet sono invarianti perché operano su distanze interatomiche scalari. PaiNN (Schütt et al., 2021) introduce **message passing equivariante** con vettori che si trasformano correttamente sotto rotazione, aggiornando sia feature scalari (invarianti) sia feature vettoriali (equivarianti) in parallelo. SEGNN (Brandstetter et al., 2021) e NequIP (Batzner et al., 2022) usano rappresentazioni tensoriali irriducibili del gruppo $SE(3)$ basate sugli armonici sferici: ogni livello l trasforma come un tensore del momento angolare l, e i prodotti tensoriali Clebsch-Gordan controllano come feature di diversi livelli interagiscono. Il costo computazionale del prodotto tensoriale irriducibile scala come $O(L^3)$ con il massimo ordine L delle armoniche — limitando spesso $L \leq 3$ nella pratica.

La **data augmentation equivariante** complementa le architetture equivarianti durante il training: se il modello è già esattamente equivariante, le augmentation di rotazione/traslazione non aggiungono informazione ma possono comunque migliorare la generalizzazione riducendo la covarianza tra esempi di training geometricamente simili. Il **Virtual Node Trick** — aggiungere un nodo artificiale connesso a tutti i nodi del grafo — permette la comunicazione globale in un singolo layer di message passing, emulando l'attenzione globale a costo lineare. Per le molecole, la combinazione di feature 3D (coordinate atomiche, angoli di legame) con feature chimiche (tipo atomico, carica) tramite architetture come SchNet o DimeNet ha prodotto potenziali interatomici di machine learning che raggiungono accuratezza ab initio a una frazione del costo computazionale.

Spectral GNN e Convoluzione sui Grafi nel Dominio di Fourier

La trasformata di Fourier sul grafo tramite autovettori laplaciani — ChebNet, GCN come approssimazione spettrale e GraphSAGE come metodo spaziale scalabile.

424.1 · Il Laplaciano del Grafo e la Trasformata di Fourier Discreta

La **convoluzione spettrale sui grafi** è motivata dall'analogia con la convoluzione euclidea: nella teoria del segnale classica, la convoluzione nel dominio spaziale corrisponde alla moltiplicazione nel dominio di Fourier. Per i grafi, il Laplaciano normalizzato simmetrico $\mathbf{L} =$

$\mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ svolge il ruolo dell'operatore differenziale, con la decomposizione spettrale $\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$ che gioca il ruolo della trasformata di Fourier: le colonne di $\mathbf{U}$ sono le "frequenze" del grafo (autovettori del Laplaciano) e gli autovalori $\mathbf{\Lambda} = \text{diag}(\lambda_1, \ldots, \lambda_n)$ con $0 = \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$ misurano la "frequenza" di ogni modo — gli autovettori a basso autovalore variano lentamente sul grafo (modi "a bassa frequenza") e quelli ad alto autovalore oscillano rapidamente tra nodi connessi. La **convoluzione spettrale** di un segnale $\mathbf{x}$ con un filtro g è: $g \star \mathbf{x} = \mathbf{U} g(\mathbf{\Lambda}) \mathbf{U}^T \mathbf{x}$ dove $g(\mathbf{\Lambda}) = \text{diag}(g(\lambda_1), \ldots, g(\lambda_n))$ è il filtro nel dominio spettrale. Il limite pratico di questa formulazione è il costo $O(n^3)$ della decomposizione spettrale e la dipendenza dalla struttura globale del grafo — non si può applicare lo stesso filtro a grafi diversi. **ChebNet** (Defferrard, Bresson e Vandergheynst 2016) approssima il filtro spettrale con polinomi di Chebyshev nel Laplaciano: $g(\mathbf{\Lambda}) \approx \sum_{k=0}^{K} \theta_k T_k(\tilde{\mathbf{\Lambda}})$ dove $\tilde{\mathbf{\Lambda}} = 2\mathbf{\Lambda}/\lambda_{\max} - \mathbf{I}$ è il Laplaciano scalato in $[-1, 1]$ e T_k sono i polinomi di Chebyshev. Questo riduce il costo a $O(K|E|)$ e produce filtri con supporto locale esattamente K-hop. GCN (Kipf e Welling 2017) semplifica ulteriormente con $K = 1$ e una normalizzazione spettrale specifica, ottenendo il famoso aggiornamento $\mathbf{H}^{(k+1)} = \sigma(\tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}\mathbf{H}^{(k)}\mathbf{W}^{(k)})$ dove $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ include self-loops.

424.2 · Filtri Polinomiali e Approssimazione Spettrale

I filtri spettrali definiti su grafi possono essere approssimati da polinomi della matrice laplaciana per evitare la costosa decomposizione agli autovalori. ChebConv (Defferrard et al., 2016) usa polinomi di Chebyshev $T_k(\tilde{L})$ della Laplaciana normalizzata $\tilde{L}$: la convoluzione diventa $g_\theta * x = \sum_{k=0}^{K} \theta_k T_k(\tilde{L}) x$, dove ogni termine richiede solo una moltiplicazione matrice-vettore con la Laplaciana sparsa — costo $O(K|E|)$ dove $|E|$ è il numero di archi. GCN (Kipf e Welling, 2017) è il caso speciale con $K = 1$ e approssimazione lineare: $\hat{A} = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$ dove $\tilde{A} = A + I$ include self-loops e $\tilde{D}$ è la matrice di grado normalizzata. Questa semplicità rende GCN estremamente efficiente ma limita la ricettività a un singolo hop per layer, richiedendo profondità crescente per catturare informazioni a lungo raggio con il rischio di over-smoothing. APPNP (Klicpera et al., 2019) separa la trasformazione delle feature (MLP locale) dalla propagazione (schema approssimato PageRank), permettendo una diffusione a lungo raggio controllata dal tasso di teletrasporto α senza il classico problema di over-smoothing dei GCN profondi.

Il Framework del Meta-apprendimento: Imparare ad Apprendere

Il task di secondo livello che trasforma l'apprendimento in un problema di ottimizzazione — episodi, support set, query set e le tre famiglie di approcci.

425.1 · Il Problema del Meta-apprendimento

Il **meta-apprendimento** affronta una limitazione fondamentale del machine learning standard: i modelli addestrati su molti dati di un task specifico generalizzano male a nuovi task con pochi dati. Un essere umano che conosce migliaia di volti riesce a riconoscere un volto nuovo dopo averlo visto una sola volta; un classificatore di immagini addestrato su ImageNet tipicamente richiede centinaia di esempi per adattarsi a una nuova categoria. Il meta-apprendimento formalizza l'obiettivo di "imparare ad apprendere": si dispone di una distribuzione di task $p(\mathcal{T})$, e si vuole trovare un algoritmo di apprendimento $\mathcal{A}$ che, dato un piccolo insieme di esempi (il *support set* $\mathcal{S}$) di un nuovo task $\mathcal{T} \sim p(\mathcal{T})$, produca velocemente una buona ipotesi che generalizza sul *query set* $\mathcal{Q}$ dello stesso task. L'ottimizzazione avviene al livello del meta-parametro: si cerca θ tale che $\mathcal{A}(\theta, \mathcal{S}_\mathcal{T})$ sia buono su $\mathcal{Q}_\mathcal{T}$ per ogni $\mathcal{T} \sim p(\mathcal{T})$. Le tre famiglie principali di approcci differiscono nel modo in cui parametrizzano $\mathcal{A}$. Le **inizializzazioni meta-apprese** (MAML, §426) cercano parametri iniziali da cui pochi passi di gradient descent producono buone ipotesi per qualunque task. Le **reti metriche** (Prototypical Networks, §427) apprendono uno spazio di embedding dove la classificazione è risolta da semplici misure di distanza. Le **hypernetworks meta-apprese** (§428) apprendono una rete che genera i parametri di una rete più piccola dato il support set.

425.2 · Episodic Training e lo Spazio dei Task

Il paradigma fondamentale del meta-apprendimento è l'**episodic training**: ogni iterazione di training consiste in un episodio che simula la struttura del problema test. Per il few-shot N-way K-shot, ogni episodio campiona N classi dallo spazio delle classi di training, per ognuna K esempi come support set e alcuni esempi come query set — il modello deve classificare correttamente le query usando solo il support. Questa struttura obbliga il modello ad apprendere un induttivo di generalizzazione all'interno di ogni episodio, anziché sfruttare la statistica globale del dataset. La distribuzione $p(\mathcal{T})$ su cui è definito il meta-apprendimento deve essere abbastanza ricca da coprire il dominio di test: se i task di training e test provengono da distribuzioni molto diverse (cross-domain few-shot), le prestazioni degradano significativamente, indicando che il meta-learner ha imparato bias specifici del dominio piuttosto che un principio di apprendimento universale. Il **task augmentation** — generare task artificiali modificando le feature o le etichette dei task esistenti — e il **meta-training multi-dominio** (STARTUP, tri-M) mitigano questo problema espandendo artificialmente la diversità dello spazio dei task durante il training.

La **meta-test evaluation** usa tipicamente benchmark standard: miniImageNet (100 classi, 600 immagini per classe, split 64/16/20 per meta-train/val/test), tieredImageNet (organizzato

per categorie semantiche, riduce il leak di informazione tra train e test), CUB-200 (uccelli, per valutare la specializzazione fine-grained) e cross-domain benchmark come Meta-Dataset che combina 10 dataset eterogenei. I risultati dipendono fortemente dal backbone: con ResNet-12 o ViT pre-addestrato su ImageNet, le differenze tra algoritmi di meta-apprendimento si riducono significativamente rispetto a backbone leggeri — suggerendo che la qualità delle feature pre-addestrate domina l'algoritmo di meta-apprendimento per dataset sufficientemente correlati.

MAML: Model-Agnostic Meta-Learning

Trovare un'inizializzazione universalmente adattabile — la derivazione del bi-level optimization, il gradient attraverso il gradient e le varianti computazionalmente efficienti.

426.1 · La Formulazione Bi-Level di MAML

MAML (Model-Agnostic Meta-Learning) (Finn, Abbeel e Levine 2017, ICML Best Paper) ottimizza i meta-parametri θ come un'inizializzazione "universalmente buona" da cui pochi passi di gradient descent su qualunque task specifico producono buone prestazioni. Per ogni task $\mathcal{T}_i$ campionato da $p(\mathcal{T})$, si esegue un'*inner update* di k passi di gradient descent sul support set: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}^{\mathcal{S}}(\theta)$ (per $k=1$). La *outer update* aggiorna θ minimizzando la loss sui query set dopo gli inner update:

$$\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}^{\mathcal{Q}}(\theta_i')$$

Il gradiente dell'outer update richiede la differenziazione attraverso le operazioni di gradient descent dell'inner update: $\nabla_\theta \mathcal{L}^{\mathcal{Q}}(\theta') = \nabla_{\theta'} \mathcal{L}^{\mathcal{Q}} \cdot \frac{\partial \theta'}{\partial \theta} = \nabla_{\theta'} \mathcal{L}^{\mathcal{Q}} \cdot (I - \alpha \nabla_\theta^2 \mathcal{L}^{\mathcal{S}})$ — il gradiente del gradiente, ovvero la matrice Hessiana della loss dell'inner update. Questo "gradient through gradient" è computazionalmente costoso (richiede differenziazione di secondo ordine) ma permette all'algoritmo di ottimizzare esplicitamente per la *adattabilità* del modello, non solo per le sue prestazioni iniziali. **First-Order MAML (FOMAML)** e **Reptile** (Nichol, Achiam e Schulman 2018) approssimano il secondo ordine con il primo, riducendo il costo computazionale: Reptile, in particolare, semplicemente media i parametri aggiornati degli inner loop invece di calcolare il gradiente esatto, con prestazioni simili a MAML a costo molto ridotto. La connessione con il continual learning è profonda: MAML applicato in sequenza a nuovi task senza accesso

ai dati passati corrisponde a una forma di meta-apprendimento continuo dove l'inizializzazione ottimale serve come meccanismo anti-forgetting.

426.2 · Varianti di MAML: FOMAML, ANIL e Meta-SGD

Il costo computazionale di MAML è dominato dal calcolo del **gradiente di secondo ordine** attraverso i passi di adattamento: lo Hessian dell'obiettivo inner-loop rispetto ai parametri iniziali richiede $O(kn)$ memoria aggiuntiva per backpropagation attraverso k gradient steps con modello da n parametri. **FOMAML** (First-Order MAML) rimuove completamente i termini di secondo ordine, approssimando il meta-gradiente con il gradiente di primo ordine calcolato ai parametri adattati θ': sorprendentemente, le prestazioni empiriche si degradano solo marginalmente rispetto al MAML esatto su molti benchmark. **Reptile** (Nichol et al., 2018) offre un'approssimazione ancor più semplice: si eseguono k passi SGD sul task, poi si aggiorna l'inizializzazione verso i parametri adattati con $\theta \leftarrow \theta + \epsilon(\theta' - \theta)$ — senza nemmeno valutare un query set separato. **ANIL** (Almost No Inner Loop, Raghu et al., 2020) mostra che la maggior parte dell'adattamento avviene nell'ultimo layer: eseguendo l'inner-loop solo sull'head di classificazione (parametri dell'ultimo layer) mentre il body rimane fisso durante l'adattamento, si ottengono prestazioni comparabili a MAML a una frazione del costo. **Meta-SGD** (Li et al., 2017) meta-apprende anche il learning rate dell'inner loop come vettore di parametri α, uno per dimensione, permettendo tassi di apprendimento anisotropi che si adattano alla geometria del loss landscape.

Prototypical Networks e il Few-Shot come Classificazione Metrica

Rappresentare ogni classe con il suo prototipo — la distanza euclidea come classificatore, la connessione con i mixture model e Matching Networks.

427.1 · La Logica dei Prototipi

Le **Prototypical Networks** (Snell, Sohl-Dickstein e Salakhutdinov 2017) applicano il principio della classificazione per distanza al few-shot learning: ogni classe c è rappresentata dal suo *prototipo* $\mathbf{p}_c$ — la media delle rappresentazioni neurali degli esempi di support per quella classe: $\mathbf{p}_c = \frac{1}{|S_c|} \sum_{(\mathbf{x}_i, y_i) \in S_c} f_\phi(\mathbf{x}_i)$ dove $f_\phi : \mathbb{R}^d \to \mathbb{R}^m$ è un encoder neurale con parametri ϕ. La classificazione di un nuovo esempio $\mathbf{x}$ usa la distanza softmax nello spazio di embedding: $P(y = c|\mathbf{x}) = \frac{\exp(-d(f_\phi(\mathbf{x}), \mathbf{p}_c))}{\sum_{c'} \exp(-d(f_\phi(\mathbf{x}), \mathbf{p}_{c'}))}$ dove d è tipicamente la distanza euclidea o la distanza coseno. Il training è episodico: ad ogni episodio si campionano N classi, ognuna con K esempi di support

e Q esempi di query, si calcolano i prototipi, e si ottimizza la cross-entropy sugli esempi di query. La semplicità di Prototypical Networks è la sua forza: il prototipo è l'estimatore di massima verosimiglianza della media di un mixture model gaussiano isotropo, e la softmax sulla distanza euclidea corrisponde alla posterior bayesiana. La **connessione con i mixture model** (Snell e Zemel 2021) mostra che Prototypical Networks è un caso speciale di variational inference su un mixture model nel spazio di embedding: il prototipo stima il centro del cluster gaussiano per ogni classe, e l'encoder apprende uno spazio dove le classi sono separabili da gaussiane isotrope.

427.2 · Reti Relazionali, Matching Networks e Distanza Appresa

Le **Matching Networks** (Vinyals et al., 2016) precedono le Prototypical Networks e introducono un meccanismo di attenzione contestuale: invece di un embedding fisso, l'encoder della query usa full-context embedding con LSTM e attenzione sul support set, permettendo all'embedding di ogni esempio di adattarsi in base al contesto degli altri esempi nel support. Il meccanismo di predizione è un'attenzione softmax sulle etichette del support ponderata dalla similarità coseno tra query e support: $\hat{y} = \sum_{i=1}^{NK} a(\hat{x}, x_i) y_i$. Le **Relation Networks** (Sung et al., 2018) vanno oltre le distanze metriche prefissate: la distanza tra un esempio di query x_q e il prototipo della classe c è predetta da una rete di relazione $g_\phi([x_q, c])$ addestrata end-to-end, permettendo metriche di confronto non simmetriche e dipendenti dal contenuto. Il **DeepEMD** (Zhang et al., 2020) usa Earth Mover's Distance (EMD) come metrica: confronta le patch spaziali di due immagini con una distanza di trasporto ottimale, catturando la struttura spaziale fine-grained delle feature. La scelta della distanza metrica rimane un iperparametro attivo di ricerca: distanza coseno, euclidea, Mahalanobis appresa e distanze basate su kernel producono risultati diversi a seconda del dominio applicativo.

Il **transfer learning come meta-apprendimento implicito** sfida la necessità di algoritmi espliciti: pre-addestrare un backbone su ImageNet e aggiungere un classificatore lineare sul support set produce risultati competitivi con i meta-learner espliciti su molti benchmark. Tian et al. (2020) dimostrano che la qualità delle feature spiega la maggior parte della varianza nelle prestazioni few-shot. Questa osservazione ha spostato il focus verso i **foundation model**: CLIP, DINO e DINOv2 producono feature così ricche che la few-shot classification diventa quasi banale con nearest-neighbor nell'embedding space — rendendo il meta-apprendimento metrico meno rilevante per i task visivi standard ma ancora fondamentale per domini specializzati (biologia, chimica, robotica) dove i foundation model pre-addestrati su ImageNet non forniscono feature direttamente trasferibili.

Hypernetworks, Meta-SGD e Amortized Optimization

Generare i parametri di una rete da un'altra rete — la relazione tra hypernetwork e meta-apprendimento, Meta-SGD e NAS come hypernetwork.

428.1 · Hypernetworks: Reti che Generano Parametri

Un **hypernetwork** (Ha, Dai e Le 2017) è una rete neurale h_ϕ che genera i parametri di una seconda rete neurale (la *target network*) f_θ: $\theta = h_\phi(\mathbf{z})$ dove $\mathbf{z}$ è un input al hypernetwork (un embedding del task, un'immagine, un prompt). Nel contesto del meta-apprendimento, l'hypernetwork riceve il support set come input e genera i parametri della classificatore adatti al task corrente: $\theta_\mathcal{T} = h_\phi(\mathcal{S}_\mathcal{T})$. Questo approccio, detto *amortized optimization*, sostituisce l'ottimizzazione iterativa degli inner loop di MAML con una singola passata forward dell'hypernetwork — molto più veloce in inferenza. Il prezzo è che l'hypernetwork deve "imparare" la funzione di ottimizzazione, il che richiede di averla vista sufficientemente in training. **Meta-SGD** (Li et al. 2017) è un'estensione di MAML che apprende non solo l'inizializzazione θ ma anche il learning rate α per ogni parametro come vettore meta-appreso, ottenendo un ottimizzatore precondizionato adattivo senza calcolo della Hessiana. La connessione con la **Neural Architecture Search (NAS)** è diretta: DARTS (Liu et al. 2019) parametrizza l'architettura come parametri continui α ottimizzati al livello outer, mentre i pesi della rete sono ottimizzati al livello inner — esattamente la struttura bi-level di MAML con θ sostituito dall'architettura α.

428.2 · Neural Processes e Inferenza Ammortizzata

I **Neural Processes** (Garnelo et al., 2018) combinano meta-apprendimento e inferenza bayesiana: l'obiettivo è apprendere una distribuzione su funzioni che si aggiorna rapidamente dato un piccolo support set senza ottimizzazione esplicita. Un NP ha tre componenti: un encoder che comprime ogni coppia support (x_i, y_i) in una rappresentazione latente r_i, un'aggregazione globale $r = \bigoplus_i r_i$ e un decoder che predice y per nuove query condizionato a r e alla query x. Gli **Attentive Neural Processes** (ANP, Kim et al., 2019) sostituiscono l'aggregazione media con cross-attention: la rappresentazione della query attende selettivamente sulle coppie support più rilevanti, producendo adattamento context-dependent senza meta-gradiente. I **Latent Neural Processes** aggiungono una variabile latente $z \sim q_\phi(z|S)$ campionata dal support con un encoder stocastico, permettendo di esprimere incertezza epistemica sulla funzione sottostante oltre all'incertezza aleatoria. L'**amortized optimization** è il principio generale: invece di ottimizzare separatamente per ogni task, si addestra una rete ad ammortizzare la soluzione sull'intera distribuzione di task — trasformando l'inferenza da un problema di ottimizzazione

iterativa a una forward pass di rete neurale.

I **Gaussian Process come Neural Process** condividono la stessa struttura: entrambi definiscono distribuzioni su funzioni e aggiornano la distribuzione dato un support set. La differenza è che i GP usano kernel fissati e aggiornamento analitico (posterior esatto), mentre i NP usano encoder/decoder neurali e aggiornamento ammortizzato (approximate posterior). I NP scalano a dati ad alta dimensione dove i GP soffrono la maledizione della dimensionalità, ma a prezzo di approssimazioni nell'inferenza. Il **Convolutional Neural Process** (CNP, Gordon et al., 2019) aggiunge invarianza traslazionale all'architettura NP tramite encoder convoluzionale, particolarmente efficace per serie temporali e dati spaziali dove la struttura locale è fondamentale.

Federated Averaging e il Problema del Coordinamento

Addestrare senza centralizzare i dati — la struttura client-server, l'algoritmo FedAvg, la convergenza e la differenza con il gradient descent distribuito standard.

429.1 · Il Framework del Federated Learning

Il **Federated Learning (FL)** (McMahan, Moore, Ramage, Hampson e Agüera y Arcas 2017) è un paradigma di addestramento distribuito progettato per scenari in cui i dati non possono o non devono essere centralizzati: dispositivi mobili che non caricano i dati degli utenti sui server, ospedali che non condividono le cartelle cliniche, organizzazioni che collaborano senza rivelare dati proprietari. Il framework standard *cross-device* ha una struttura client-server: un server centrale coordina M client (tipicamente $M \sim 10^6$ dispositivi mobili) ognuno dei quali ha un dataset locale privato $\mathcal{D}_k$. L'obiettivo è minimizzare la funzione obiettivo globale $F(\theta) = \sum_{k=1}^{M} \frac{n_k}{n} F_k(\theta)$ dove $F_k(\theta) = \frac{1}{n_k} \sum_{i \in \mathcal{D}_k} \ell(\theta; x_i, y_i)$ è la loss locale del client k, $n_k = |\mathcal{D}_k|$, e $n = \sum_k n_k$. L'algoritmo **FedAvg** procede per round: il server invia i parametri correnti θ^t a un sottoinsieme casuale $\mathcal{C}$ di client; ogni client esegue E epoche di SGD locale ottenendo θ_k^{t+1}; il server aggrega con media pesata: $\theta^{t+1} = \sum_{k \in \mathcal{C}} \frac{n_k}{\sum_{j \in \mathcal{C}} n_j} \theta_k^{t+1}$. La differenza cruciale rispetto al gradient descent distribuito standard (dove ogni nodo calcola solo il gradiente e non aggiorna localmente) è il **multi-step local SGD**: ogni client esegue $E > 1$ passi di ottimizzazione locale prima di comunicare al server, riducendo il costo di comunicazione di un fattore E a spese di un bias introdotto dalla divergenza dei parametri locali.

429.2 · Sicurezza e Robustezza in Federated Learning

Il paradigma federato introduce superfici di attacco assenti nel training centralizzato. Gli attacchi di **poisoning** si dividono in data poisoning (un client corrompe il proprio dataset locale) e model poisoning (un client invia aggiornamenti malevoli costruiti ad arte). L'attacco **backdoor** (Bagdasaryan et al., 2020) inserisce una trigger pattern nei dati locali di un client corrotto: il modello aggregato mantiene il comportamento corretto sui dati normali ma produce la classe target predefinita ogni volta che incontra il trigger. La difficoltà difensiva è che gli aggiornamenti del client corrotto sono indistinguibili dagli aggiornamenti legittimi fino alla valutazione sul trigger — rendendo il rilevamento basato su norma o distanza insufficiente. L'aggregazione robusta come **Krum** (Blanchard et al., 2017) seleziona l'aggiornamento con la minima somma delle distanze ai $n - f - 2$ aggiornamenti più vicini, resistendo a $f < n/2$ client corrotti. **FLTrust** (Cao et al., 2021) mantiene un piccolo dataset di riferimento pulito sul server e assegna a ogni aggiornamento client un trust score proporzionale alla similarità coseno con il gradiente di riferimento — downpesando automaticamente gli aggiornamenti anomali. La tensione tra privacy (no accesso ai dati client) e robustezza (serve qualche forma di supervisione) rimane un problema aperto fondamentale nel federated learning sicuro.

La **differential privacy in practice** per il federated learning richiede la calibrazione del noise multiplier σ rispetto al gradient clipping threshold C e al privacy budget ε desiderato. Per GPT-class models su dati medici, la pratica corrente usa $\varepsilon \leq 8$ con $\delta = 10^{-5}$ — valori che producono una degradazione di accuratezza del 3–7% rispetto al training senza DP, considerata accettabile per molte applicazioni cliniche. La **secure aggregation** (SecAgg, Bonawitz et al., 2017) permette al server di computare la somma degli aggiornamenti senza vedere gli aggiornamenti individuali dei client usando protocolli crittografici basati su secret sharing — combinando privacy differenziale e secure aggregation per garanzie di privacy end-to-end.

Eterogeneità e Client Drift nel Federated Learning

Il problema della non-iidità — la divergenza dei parametri locali, il fenomeno del client drift, FedProx e SCAFFOLD.

430.1 · Il Problema della Non-Iidità

Il punto debole di FedAvg emerge quando i dati locali non sono distribuiti identicamente tra i client — la condizione **non-IID** (non Independent and Identically Distributed) che è quasi

sempre presente nelle applicazioni reali. Un utente di smartphone genera dati di un certo tipo linguistico o comportamentale; un ospedale tratta una popolazione di pazienti diversa da un altro. Quando i dati sono non-IID, l'ottimo locale $\theta_k^* = \arg\min F_k(\theta)$ può essere molto diverso dall'ottimo globale $\theta^* = \arg\min F(\theta)$, e il multi-step SGD locale porta i parametri verso il proprio ottimo locale invece che verso quello globale — il fenomeno del **client drift**. L'effetto pratico è che FedAvg con dati non-IID può divergere o convergere molto lentamente, soprattutto con E grande. **FedProx** (Li, Sahu, Talwalkar e Smith 2020) mitiga il drift aggiungendo un termine di prossimalità alla loss locale: $h_k(\theta; \theta^t) = F_k(\theta) + \frac{\mu}{2}\|\theta - \theta^t\|^2$ che penalizza la distanza dei parametri locali da quelli del server. Il coefficiente μ controlla il tradeoff tra ottimizzazione locale e coesione globale: $\mu = 0$ recupera FedAvg, $\mu \to \infty$ forza ogni client a non spostarsi dall'inizializzazione del server. **SCAFFOLD** (Karimireddy et al. 2020) usa variabili di controllo per stimare e correggere il drift direttamente nel gradiente locale, ottenendo convergenza teorica senza la penalità di prossimalità.

430.2 · Personalized Federated Learning e Clustered Federated Learning

Il **Personalized Federated Learning** riconosce che un singolo modello globale potrebbe essere sub-ottimale per ogni client individuale: l'obiettivo non è convergere a un unico w^* ma imparare una famiglia di modelli $\{w_k^*\}$ personalizzati che bilancino adattamento locale e conoscenza condivisa. **pFedMe** (Dinh et al., 2020) formula l'obiettivo come $\min_{w,w_1,\ldots,w_K} \sum_k F_k(w_k^*; w)$ dove $w_k^* = \arg\min_{w_k} F_k(w_k) + \frac{\lambda}{2}\|w_k - w\|^2$: ogni client personalizza il modello locale vincolato a restare vicino al modello globale. **IFCA** (Ghosh et al., 2020) partiziona i client in C cluster, dove ogni cluster condivide un modello: i client vengono assegnati al cluster con la loss minore iterativamente, alternando stima dell'assegnazione e aggiornamento del modello per ogni cluster. Il **Federated Meta-Learning** combina MAML con federated averaging: il server aggrega i meta-gradienti dai client per aggiornare l'inizializzazione globale, che ogni client può adattare localmente con pochi passi gradient — producendo un inizializzazione meta-appresa distribuita senza condividere dati locali.

La **personalizzazione locale** può anche avvenire tramite *fine-tuning post-aggregazione*: dopo che il modello globale converge, ogni client esegue pochi passi di gradient descent sul proprio dataset locale producendo un modello personalizzato $w_k = w_{global} - \eta \nabla F_k(w_{global})$. Questo approccio Moreau-envelope (pFedMe) o meta-learning (Per-FedAvg) esplicita il tradeoff tra generalizzazione globale e adattamento locale tramite un iperparametro che controlla la distanza dall'inizializzazione globale. Le applicazioni reali — tastiere predittive su smartphone, modelli medici multi-ospedale — mostrano che il guadagno della personalizzazione rispetto al modello globale cresce con l'eterogeneità dei dati, confermando l'utilità del personalized FL sui

benchmark standard come FEMNIST e Shakespeare.

Compressione della Comunicazione nel Federated Learning

Il collo di bottiglia della larghezza di banda — quantizzazione dei gradienti, sketching, top-k sparsification e l'impatto sulla convergenza.

431.1 · Il Costo della Comunicazione

In un sistema FL cross-device con modelli da miliardi di parametri, comunicare l'intero vettore di gradienti a ogni round è proibitivo: un modello da 1 miliardo di parametri in float32 richiede 4 GB per round, inaccettabile per dispositivi mobili su connessioni mobili. Le tecniche di **compressione della comunicazione** riducono questo costo di 100x–1000x. La **quantizzazione dei gradienti** rappresenta ogni componente del gradiente con pochi bit invece di 32: QSGD (Alistarh et al. 2017) usa uno schema di quantizzazione stocastica che mantiene lo stimatore non biasato; 1-bit SGD usa solo il segno del gradiente con una correzione dell'errore di feedback. La **sparsification top-k** trasmette solo i k gradienti con valore assoluto maggiore: $\text{Top}_k(\nabla F_k) =$ i k componenti di valore assoluto più grande, con k tipicamente il 0.1–1% della dimensione totale. L'**errore di feedback** (Seide et al. 2014) accumula la parte non trasmessa del gradiente e la aggiunge al gradiente del round successivo, garantendo che nessuna informazione si perda permanentemente. La convergenza dei metodi con compressione richiede la condizione di *riduzione contratta*: $\mathbb{E}[\|C(\mathbf{g})\|^2] \geq (1-\delta)\|\mathbf{g}\|^2$ per qualche $\delta \in (0,1)$, che assicura che l'operatore di compressione C mantenga almeno una frazione $(1-\delta)$ dell'informazione originale.

431.2 · Comunicazione Asincrona e Fault Tolerance

Nell'aggregazione sincrona (FedAvg standard), il server attende tutti i client prima di aggiornare il modello globale — un **straggler** lento blocca l'intero round. L'aggregazione **asincrona** permette al server di aggiornare il modello globale non appena riceve K aggiornamenti da un qualsiasi sottoinsieme di client, ignorando i client lenti. Il problema dell'asincronicità è la **staleness**: un client che calcola il gradiente su una versione t-step-vecchia del modello invia un aggiornamento potenzialmente outdated. **FedBuff** (Nguyen et al., 2022) gestisce la staleness con un buffer aggregato: raccoglie aggiornamenti da più client e li applica insieme, bilanciando latenza e staleness. **SCAFFOLD asincrono** e varianti con pesi di aggregazione dipendenti dalla staleness $w_k \propto (1+\tau_k)^{-\alpha}$ (dove τ_k è il ritardo) garantiscono convergenza teorica. La

resilienza ai Byzantine clients — client che trasmettono aggiornamenti malevoli — richiede aggregazione robusta: Krum (Blanchard et al., 2017) seleziona l'aggiornamento più simile agli altri $n - f - 2$ vicini, resistendo a f client corrotti su n totali; Trimmed Mean rimuove le code della distribuzione degli aggiornamenti prima di aggregare.

La **comunicazione asincrona** introduce il problema della staleness: un client che calcola il gradiente su una versione τ-step-vecchia del modello invia un aggiornamento potenzialmente outdated. L'impatto sulla convergenza dipende dal gradiente Lipschitz: per funzioni L-smooth, l'errore di staleness τ introduce un termine additivo $O(\tau^2\eta^2 L)$ nel bound di convergenza. FedBuff (Nguyen et al., 2022) mitiga il problema bufferizzando K aggiornamenti asincroni prima di applicarli, riducendo la varianza della staleness a costo di una latenza moderata. La misurazione della **fault tolerance** nei sistemi federati usa il modello Byzantine fault tolerance: un sistema con n client è f-Byzantine-tolerant se mantiene le garanzie di convergenza con al più f client corrotti, richiedendo tipicamente $f < n/3$ con aggregazione robusta.

Federated Learning con Privacy Differenziale

Proteggere la privacy degli utenti nell'aggregazione — DP-SGD, la composizione del privacy budget, la secure aggregation e i tradeoff pratica-teoria.

432.1 · Privacy Differenziale nel Federated Learning

Il Federated Learning riduce l'esposizione dei dati non trasmettendoli al server, ma i modelli aggiornati possono ancora rivelare informazioni sui dati di training tramite attacchi di inversione del modello (Fredrikson et al. 2015) o attacchi di membership inference (Shokri et al. 2017). La **privacy differenziale (DP)** (§228) fornisce garanzie formali contro questo: un meccanismo $\mathcal{M}$ è (ε, δ)-differenzialmente privato se per qualunque dataset adiacente D, D' (differenti per un singolo utente) e qualunque output S: $P(\mathcal{M}(D) \in S) \leq e^{\varepsilon} P(\mathcal{M}(D') \in S) + \delta$. L'integrazione di DP nel FL avviene a due livelli. **Local DP** (LDP): ogni client aggiunge rumore ai propri aggiornamenti prima di trasmetterli al server, garantendo che nessun avversario che osserva la comunicazione possa inferire i dati locali. Il meccanismo standard è il **Gaussian mechanism**: si clippa la norma dell'aggiornamento a C (clipping del gradiente) e si aggiunge rumore gaussiano $\mathcal{N}(0, \sigma^2 C^2 I)$. **Central DP**: il server aggiunge rumore all'aggregato dopo averlo ricevuto dai client, richiedendo che il server sia onesto. Il **privacy budget** si esaurisce ad ogni round: per il **moment accountant** (Abadi et al. 2016), dopo T round la privacy totale è (ε, δ) con $\varepsilon = O(q\sqrt{T \log(1/\delta)}/\sigma)$ dove q è la frazione di client partecipanti ad ogni round. C'è un trade-

off fondamentale: più rumore si aggiunge (maggiore σ, maggiore privacy), minore è l'utilità del modello. I migliori risultati empirici si ottengono con $\varepsilon \approx 8$–10 su benchmark standard come CIFAR-10, il che offre protezione significativa contro molti attacchi pratici a un costo di accuratezza del 2–5%.

432.2 · Garanzie Formali di Privacy: Meccanismo Gaussian e Composizione

La **privacy differenziale (DP)** formalizza la garanzia che un algoritmo non riveli informazioni specifiche su singoli individui: un algoritmo $\mathcal{M}$ soddisfa (ε, δ)-DP se per qualsiasi coppia di dataset adiacenti D, D' che differiscono in un singolo record e qualsiasi output S: $P[\mathcal{M}(D) \in S] \leq e^{\varepsilon} P[\mathcal{M}(D') \in S] + \delta$. In Federated Learning con DP, ogni client aggiunge rumore ai propri aggiornamenti prima di trasmetterli. Il **meccanismo Gaussian** aggiunge rumore $\mathcal{N}(0, \sigma^2 S_f^2 \cdot I)$ dove S_f è la sensibilità L_2 della query (controllata dal gradient clipping $\|g\|_2 \leq C$) e σ è calibrato per soddisfare (ε, δ)-DP. Il **teorema di composizione forte** (Dwork et al., 2010) e il moments accountant (Abadi et al., 2016) permettono di calcolare il budget privacy consumato da molti round di comunicazione: la perdita di privacy cresce come $O(\sqrt{T})$ con il numero di round T, richiedendo un tradeoff esplicito tra numero di round, rumore aggiunto e accuratezza finale. La **local DP** — dove ogni client aggiunge rumore ai propri dati prima ancora di calcolare il gradiente — offre garanzie più forti (il server non vede mai i dati grezzi) ma richiede rumore molto maggiore per la stessa ε, producendo degrado di accuratezza sostanziale.

La **privacy amplification by sampling** è una tecnica che riduce il budget ε sfruttando il subsampling: se ogni round di FL usa solo una frazione q dei client, le garanzie di privacy migliorano come $\varepsilon' \approx q\varepsilon$ (amplification lemma). Combinando subsampling, clipping e noise aggiunto, il **moments accountant** traccia con precisione il budget ε consumato attraverso T round: il trade-space $(\varepsilon, T, \sigma, C, q)$ consente di ottimizzare la qualità del modello finale sotto vincoli di privacy dati. I benchmark DP-FL come DP-CIFAR e DP-Medical mostrano che la gap di accuratezza rispetto al training non-privato si chiude rapidamente all'aumentare della dimensione del dataset aggregato — sottolineando il vantaggio del federated approach per raccogliere dati sufficienti a compensare il rumore differenziale.

Circuit Analysis nei Transformer: Scomporre i Calcoli in Circuiti

La prospettiva dei circuiti per i transformer — composizione degli head, il residual stream e la metodologia di individuazione dei circuiti.

433.1 · La Visione dei Circuiti nei Transformer

L'**interpretabilità meccanicistica** (Olah et al. 2020, Anthropic 2022, DeepMind 2022) è il programma di ricerca che tenta di comprendere come i transformer implementano i loro calcoli a livello di componenti specifici — attenzione, MLP, layer normalization — invece di trattarli come scatole nere. La prospettiva centrale è quella dei **circuiti**: subset di componenti del modello che collaborano per implementare un calcolo specifico. Nel framework del **residual stream** di Elhage et al. (2021), ogni layer del transformer *legge* dal residual stream $\mathbf{x}^{(l)}$ tramite le sue matrici di proiezione e *scrive* nel residual stream tramite addizione: $\mathbf{x}^{(l+1)} = \mathbf{x}^{(l)} + \mathrm{MHAttn}^{(l)}(\mathbf{x}^{(l)}) + \mathrm{MLP}^{(l)}(\mathbf{x}^{(l)} + \mathrm{MHAttn}^{(l)}(\mathbf{x}^{(l)}))$. Questa struttura additiva fa sì che il residual stream $\mathbf{x}^{(l)}$ sia una *sovrapposizione lineare* di contributi di tutti i layer precedenti, rendendo la composizione dei contributi tracciabile. Il principio di **superposition** (Elman 1991, Hinton 2022) afferma che le reti neurali rappresentano più *feature* di quanto la loro dimensione nominale suggerirebbe, usando direzioni quasi-ortogonali per codificare molte feature su pochi neuroni — una forma di compressione con informazione che si ricostruisce tramite la quasi-ortogonalità.

433.2 · La Metodologia di Individuazione dei Circuiti

La metodologia standard per identificare un circuito che implementa un task specifico si articola in tre fasi. Prima, si identifica un **dataset diagnostico** dove il comportamento del modello è semplice e controllabile (ad esempio, il task di "indirect object identification" di Wang et al. 2022: dato "John e Mary andarono al negozio; John diede un libro a $''$,ilmodellodevecompletarecon"Mary$''$).Seconda,siusal'$\mathbf{acti}$

433.2 · Superposizione e Polisemia nelle Reti Neurali

Una delle scoperte più sorprendenti dell'interpretabilità meccanicistica è il fenomeno della **superposizione** (Ely e Bricken, 2022): le reti neurali rappresentano più feature di quante abbiano neuroni, "comprimendo" feature quasi-ortogonali nello stesso spazio di attivazione. Se le feature attive sono sparse — poche per ogni input — la sovrapposizione interferisce raramente e la rete recupera le feature individuali grazie alla sparsità. Formalmente, una rete con n neuroni può rappresentare $m \gg n$ feature con errore $O(m/n^2)$ usando frame di equiangular tight frames (ETF), dove ogni feature corrisponde a un vettore in $\mathbb{R}^n$ con correlazioni controllate. La **polisemia dei neuroni** — singoli neuroni che si attivano per concetti semanticamente diversi — è una conseguenza della superposizione: il neurone 2049 di GPT-2 si attiva per il pronome inglese "I", per concetti in russo e per numeri decimali. Gli **sparse autoencoder** (Cunningham et al., 2023) affrontano la polisemia addestrando un autoencoder con regularizzazione L_1 sulle attivazioni intermedie del modello: le feature dello spazio latente allargato (tipicamente $4imes$–$16imes$ il layer originale) risultano monosemantiche e interpretabili, permettendo una

mappa delle concept features molto più leggibile dei neuroni originali.

Funzioni degli Attention Head: Tassonomia e Universalità

Che cosa fanno veramente le teste di attenzione — copia, spostamento, ricerca in tabella, e i motivi ricorrenti attraverso scale diverse.

434.1 · La Composizione delle Teste di Attenzione

Un'attention head in un transformer può essere caratterizzata analizzando la struttura a basso rango delle sue matrici W_Q, W_K, W_V, W_O. La matrice $W_QK = W_Q^T W_K$ determina quali coppie posizione-token vengono associate; la matrice $W_{OV} = W_O W_V$ determina come il contenuto dei token sorgente viene scritto nella posizione destinataria. Olah et al. (2020) e Elhage et al. (2021) hanno catalogato diversi **tipi funzionali** di attention head. Le **copying heads** hanno W_{OV} approssimativamente uguale all'identità, copiando i token direttamente dalla posizione sorgente a quella destinataria — usate per il "look-ahead" e il completamento di pattern ripetuti. Le **previous-token heads** attendono quasi sempre al token immediatamente precedente — usate per trasmettere informazioni locali. Le **induction heads** (illustrate sopra) implementano la lookup in un pattern di memoria implicita. Le **fuzzy induction heads** generalizzano le induction heads al caso in cui il "pattern" non deve essere esattamente uguale ma semanticamente simile — consentendo il "few-shot in-context learning" secondo Olsson et al. (2022). L'**universalità** delle induction heads è stata documentata empiricamente: le stesse strutture, con matrici di peso simili normalizzate, compaiono in transformer di dimensioni molto diverse addestrati su dataset diversi — suggerendo che questi circuiti sono attractor dello spazio delle reti neurali per la modellazione del linguaggio.

L'**ablation causale** distingue componenti necessari da componenti sufficienti: un componente è necessario se la sua rimozione degrada le prestazioni, sufficiente se da solo produce il comportamento target. Molti circuiti nei Transformer mostrano ridondanza — più head replicano la stessa funzione — garantendo robustezza ma complicando l'attribuzione causale. Il metodo **path patching** (Wang et al., 2023) estende l'activation patching a cammini computazionali interi nel grafo residual stream, identificando quali connessioni tra componenti trasportano informazione causalmente rilevante per il task — producendo mappe di dipendenza più precise dei metodi di ablazione puntuale.

434.2 · Ablation, Patching e Causalità Computazionale

L'identificazione della funzione di un attention head richiede tecniche di intervento causale oltre all'osservazione correlazionale. **Activation patching** (Meng et al., 2022) sostituisce le attivazioni di un componente specifico con quelle calcolate su un input di riferimento e misura il cambiamento nell'output: se patchare il componente C da un input "Roma-Italia" a un input "Berlino-Francia" cambia la predizione da "Roma" a "Berlino", allora C partecipa causalmente alla computazione del fatto geografico. L'**ablation causal tracing** raffina questa tecnica eseguendo patching su singoli layer, singole head o singoli neuroni — costruendo una mappa di causalità fine-grained del circuito. Il metodo **ACDC** (Conmy et al., 2023) automatizza la scoperta di circuiti tramite ricerca greedy: partendo dal grafico computazionale completo del modello, rimuove iterativamente i componenti il cui ablation cambia meno l'output sul task di interesse, producendo il sotto-circuito minimale necessario. I **logit lens** (Nostalgebraist, 2020) e il **tuned lens** proiettano le residual stream intermediate nello spazio del vocabolario a ogni layer, permettendo di osservare come la predizione della prossima parola si "forma" progressivamente attraverso i layer del Transformer — rivelando che le decisioni semantiche high-level avvengono nei layer intermedi mentre i layer finali affinano la distribuzione di probabilità sul vocabolario.

Probing e la Rappresentazione Lineare dell'Informazione

Cosa sanno le rappresentazioni intermedie — i classificatori di probing, il probing causale e il dibattito sull'interpretabilità del probing.

435.1 · La Metodologia del Probing

Il **probing** (Adi et al. 2017, Belinkov et al. 2017) è la tecnica più usata per analizzare le rappresentazioni intermedie dei transformer: si addestra un classificatore lineare (o una piccola rete) sull'attivazione di un layer specifico per predire una proprietà linguistica target (POS tag, dipendenze sintattiche, relazioni semantiche), e si misura la sua accuratezza. Se il classificatore di probing raggiunge alta accuratezza, si conclude che quella proprietà è linearmente codificata nella rappresentazione di quel layer. L'analisi per layer rivela come l'informazione linguistica si organizza attraverso la profondità del modello: nei BERT-like, i layer bassi tendono a codificare informazioni morfologiche e di superficie, i layer intermedi le strutture sintattiche, e i layer alti la semantica e le relazioni coreferenciali. La **ipotesi della rappresentazione lineare** (Mikolov et al. 2013, Elhage et al. 2022) afferma più ampiamente che le reti neurali codificano concetti come

direzioni lineari nello spazio di attivazione: la famosa equazione $\text{king} - \text{man} + \text{woman} \approx \text{queen}$ negli embedding di word2vec è la prima evidenza di questa struttura. Nei LLM moderni, direzioni lineari per concetti come "verità/falsità", "formalità", "positivo/negativo" sono state identificate e usate per steering del modello (Burns et al. 2022, Marks e Tegmark 2023). La critica principale al probing è che l'accuratezza del probe può riflettere l'espressività del probe stesso piuttosto che la struttura della rappresentazione: un probe lineare accurato implica codifica lineare, ma un probe non lineare accurato non implica nulla di preciso sulla struttura interna.

435.2 · Steering Vectors e Controllo delle Rappresentazioni

Se un concetto è linearmente codificato nelle attivazioni di un modello, è possibile non solo rilevarlo ma anche **modificarlo** direttamente durante l'inferenza. Gli **steering vectors** (Subramani et al., 2022; Turner et al., 2023) identificano la direzione dello spazio delle attivazioni associata a un concetto — ad esempio "onestà" vs "inganno" — e la aggiungono o sottraggono dalle attivazioni residual stream durante la forward pass: $h_l \leftarrow h_l + \alpha \cdot v_{concept}$. Il vettore $v_{concept}$ si estrae con differenza di medie su coppie contrastativi (attivazioni di frasi oneste meno attivazioni di frasi ingannevoli) o tramite PCA sulle differenze. Zou et al. (2023) applicano questa tecnica per controllare il comportamento di LLaMA su task di seguire istruzioni pericolose: aggiungere la direzione "rifiuto" riduce significativamente la compliance con richieste malevoli senza fine-tuning. **Representation Engineering** (Zou et al., 2023) generalizza il concetto a una varietà di dimensioni comportamentali — moralità, emozioni, rischio — estraendo sistematicamente le direzioni di controllo con probing contrastivo. Il **concept bottleneck probing** inserisce uno strato intermedio interpretabile nella catena computazionale: la predizione finale passa obbligatoriamente attraverso un bottleneck di concetti leggibili, permettendo sia interpretabilità sia controllo causale verificabile della logica di classificazione.

Il **Contrastive Activation Addition (CAA)** è una tecnica di steering più raffinata: anziché aggiungere il vettore di differenza media, si usa l'analisi delle componenti principali sull'insieme delle differenze di attivazione per estrarre le direzioni più stabili e ortogonali del concetto target. Questa ortonormalizzazione riduce l'interferenza con altre feature codificate nella stessa regione dello spazio delle attivazioni. Le applicazioni pratiche includono il controllo del sentiment nelle risposte, la modulazione del tono formale/informale e la riduzione dei bias demografici — tutte senza alcun fine-tuning, rendendole applicabili a modelli closed-source tramite accesso all'API se si può misurare l'output.

Causal Tracing e Localizzazione delle Conoscenze Fattuali

Dove vivono i fatti nei transformer — il metodo di causal tracing di Meng et al., gli MLP come memorie chiave-valore e il model editing.

436.1 · Il Metodo del Causal Tracing

Il **causal tracing** (Meng, Bau, Andonian e Belinkov 2022, ROME) è una tecnica per localizzare dove nel transformer viene memorizzato un fatto specifico (ad esempio, "La Torre Eiffel si trova a Parigi"). Il metodo opera con tre passi. Nel **passo pulito**, si calcola l'attivazione di ogni componente per il prompt originale ("La Torre Eiffel si trova nella città di") e si registra la risposta corretta. Nel **passo corrotto**, si sostituisce l'entità chiave con rumore ("La axjmk kfple si trova nella città di") e si registra la risposta (tipicamente errata). Nel **passo di patching**, si ripete il passo corrotto ma si "ripristina" l'attivazione di un singolo componente al suo valore pulito, misurando quanto questo ripristino riporta la risposta corretta. I componenti che causano il massimo ripristino sono quelli che "contengono" il fatto. Il risultato sorprendente di Meng et al. è che i fatti specifici tendono a essere localizzati nei layer MLP di profondità media, con la posizione del soggetto ("Torre Eiffel") come punto critico. Questo ha motivato l'algoritmo di **model editing ROME** (Rank-One Model Editing): si calcola l'aggiornamento di rango 1 alla matrice W dell'MLP identificato che cambia la rappresentazione dell'entità per associarla al nuovo fatto, senza alterare il comportamento del modello su altri input. MEMIT (Mass-Editing Memory In a Transformer) estende ROME a migliaia di edit simultanei distribuiti su più layer, con garanzie di non-interferenza tra gli edit.

436.2 · Knowledge Editing e Aggiornamento Parametrico delle Conoscenze

Il causal tracing (§436.1) ha motivato lo sviluppo di metodi per **editare le conoscenze fattuali** direttamente nei parametri del modello senza re-training completo. **ROME** (Rank-One Model Editing, Meng et al., 2022) usa il causal tracing per identificare i layer MLP dove la conoscenza è localizzata, poi applica un aggiornamento rank-one: $W_{new} = W + \Delta$ dove $\Delta = \frac{(v^* - Wk)(ck^T)}{ck^T k}$ inserisce la nuova "key-value" (k, v^*) nella matrice di proiezione del layer identificato. **MEMIT** (Mass-Editing Memory in a Transformer) estende ROME a aggiornamenti batch di migliaia di fatti distribuendo la modifica su più layer per evitare degrado. Il problema fondamentale dell'editing è la **generalizzazione della modifica**: editare "la capitale della Francia è Marsiglia" dovrebbe propagarsi a tutte le domande correlate ("dove si trova il governo francese?") senza modificare conoscenze non correlate. Benchmark come CounterFact e RIPPLE misurano questa propagazione — i metodi attuali mostrano buona localizzazione ma generalizzazione limitata, con degradazione dell'accuratezza su fatti adiacenti. Approcci alternativi come **GRACE**

usano una memoria esterna non parametrica (codebook di edits) consultata durante l'inferenza, evitando modifiche ai pesi ma aggiungendo latenza.

Il **Model Written Evaluation** (Perez et al., 2022) usa LLM per generare automaticamente dataset di valutazione per comportamenti specifici — sycophancy, manipolazione, disonestà — che sarebbero costosi da costruire manualmente. Il causal tracing combinato con l'editing parametrico apre la prospettiva di un **modello di sicurezza basato su chirurgia parametrica**: identificare i circuiti responsabili di comportamenti rischiosi tramite causal tracing e rimuoverli o modificarli tramite ROME/MEMIT — in alternativa al behavioral fine-tuning tradizionale che modifica i pesi globalmente senza garanzie meccanicistiche. Questa direzione di ricerca, pur promettente, rimane aperta: la rimozione di circuiti specifici può causare effetti collaterali imprevedibili su comportamenti correlati che condividono parte del circuito.

Quantizzazione: INT8, INT4 e GPTQ

Ridurre la precisione numerica dei pesi senza perdere accuratezza — quantizzazione post-training, calibrazione e metodi di riquantizzazione di secondo ordine.

437.1 · La Matematica della Quantizzazione

La **quantizzazione** di un modello neurale riduce la precisione numerica dei pesi (e opzionalmente delle attivazioni) da float32 o float16 a tipi interi a bassa precisione come INT8 o INT4, riducendo il costo di storage e accelerando l'inferenza su hardware che supporta operazioni intere veloci. La quantizzazione lineare uniforme di un tensore $\mathbf{W}$ a b bit usa la trasformazione: $\mathbf{W}_q = \text{round}\left(\frac{\mathbf{W}-z}{\Delta}\right)$ con $\Delta = \frac{\max(\mathbf{W})-\min(\mathbf{W})}{2^b-1}$ come scala e $z = \text{round}(-\min(\mathbf{W})/\Delta)$ come zero-point. La deantizzazione per l'inferenza recupera il tensore approssimato: $\hat{\mathbf{W}} = \Delta(\mathbf{W}_q - z) \approx \mathbf{W}$. L'errore di quantizzazione è $|\hat{\mathbf{W}} - \mathbf{W}| \leq \Delta/2$, quindi più è piccolo il range $[\min, \max]$ del tensore, minore è l'errore per bit. Nella pratica, si usa la **quantizzazione per canale** (column-wise o row-wise) dove Δ e z dipendono dalla colonna del peso, ottenendo errori più piccoli a parità di bit rispetto alla quantizzazione per tensore. La **quantizzazione a 8 bit (INT8)** produce perdite di accuratezza tipicamente inferiori all'1% rispetto al float16 con una riduzione di memoria del 2x; la **quantizzazione a 4 bit (INT4)** riduce la memoria del 4x rispetto a float16 ma richiede tecniche più sofisticate per mantenere la qualità.

437.2 · GPTQ: Quantizzazione di Secondo Ordine per i LLM

GPTQ (Frantar, Ashkboos, Hoefler e Alistarh 2022) è il metodo di *post-training quantization* (PTQ) di riferimento per i LLM, basato sull'algoritmo Optimal Brain Quantization (OBQ). L'idea è di minimizzare l'errore quadratico $\|(\mathbf{W}-\hat{\mathbf{W}})\mathbf{X}\|_F^2$ dove $\mathbf{X}$ è la matrice di attivazione su un piccolo set di calibrazione, quantizzando un peso alla volta e compensando l'errore introdotto aggiornando i pesi rimanenti. La soluzione di secondo ordine usa la matrice Hessiana della loss rispetto ai pesi $H = 2\mathbf{X}\mathbf{X}^T$: quando si quantizza il peso w_q, l'aggiornamento dei pesi rimanenti che compensa l'errore è $\delta \mathbf{W}_F = -\frac{w_q - \hat{w}_q}{H^{-1}_{qq}} \cdot (H^{-1})_{:,q} \cdot e_q^T$ dove H^{-1}_{qq} è l'entrata diagonale dell'inversa Hessiana. GPTQ quantizza i pesi colonna per colonna (lazy batch update) con una singola passata in $O(d \cdot k^2)$ invece di $O(d^3)$, rendendo fattibile la quantizzazione di modelli con miliardi di parametri in poche ore su una singola GPU. Il risultato pratico è impressionante: LLaMA-2 7B quantizzato a INT4 con GPTQ occupa circa 3.5 GB ed è eseguibile su un laptop consumer, con una perdita di perplexity di meno del 5% rispetto al modello float16.

437.3 · Quantizzazione Mista e Deployment su Hardware Eterogeneo

La **quantizzazione mista** (mixed-precision quantization) assegna bit-width diverse a layer o tensori diversi in base alla loro sensibilità: layer critici per la qualità (spesso i primi e gli ultimi della rete, e le attention weights) usano INT8 o FP16, layer meno sensibili usano INT4 o INT2. HAQ (Wang et al., 2019) usa reinforcement learning per determinare automaticamente la bit-width ottimale per ogni layer dato un budget di risorse; GPTQ e AWQ operano su LLM calibrandosi su pochi campioni del dataset senza richiedere re-training. L'**W4A16 quantization** — pesi INT4, attivazioni FP16 — è diventato lo standard de facto per l'inferenza di LLM su GPU consumer: i pesi occupano 4*imes* meno memoria rispetto a FP16, riducendo la pressione di memoria che è il collo di bottiglia dominante per il decoding autoregressivo, mentre il calcolo avviene comunque in FP16 per mantenere la qualità numerica. La quantizzazione sub-4-bit (INT2, INT3) richiede tecniche di **groupwise quantization**: invece di una scala globale per tensore, si usa una scala separata per ogni gruppo di g elementi (tipicamente $g = 128$), aumentando leggermente l'overhead di memoria ma preservando precisione sufficiente. Il formato **NF4 (Normal Float 4)** usato da QLoRA è un caso speciale di quantizzazione ottimale per distribuzioni normali: i livelli di quantizzazione sono posizionati in modo da minimizzare l'errore quadratico atteso quando i valori seguono una distribuzione gaussiana, ottenendo qualità superiore a INT4 uniforme.

Pruning: Rimuovere i Pesi Ridondanti

Trovare e rimuovere i pesi meno importanti — pruning non strutturato con l'Optimal Brain Surgeon, pruning strutturato per head e layer, e la Lottery Ticket Hypothesis.

438.1 · Pruning Non Strutturato e di Secondo Ordine

Il **pruning** di una rete neurale rimuove i pesi (o neuroni, o layer) meno importanti per ridurre il costo computazionale e di storage. Il **pruning non strutturato** rimuove pesi singoli, producendo matrici sparse; il **pruning strutturato** rimuove interi canali, attention head, o layer, producendo una rete più piccola con architettura regolare eseguibile efficientemente su hardware standard. Il criterio più semplice è il **magnitude pruning**: si rimuovono i pesi con valore assoluto più piccolo. Nonostante la sua semplicità, funziona sorprendentemente bene quando seguito da fine-tuning. Il metodo di secondo ordine **Optimal Brain Surgeon (OBS)** (LeCun et al. 1990, Hassibi e Stork 1993) usa la Hessiana della loss per stimare l'impatto della rimozione di ogni peso: il cambio di loss causato dalla rimozione del peso w_i è $\Delta L_i \approx \frac{w_i^2}{2(H^{-1})_{ii}}$ dove H è la Hessiana della loss. Si rimuove il peso con ΔL_i minimo e si compensano gli altri pesi: $\delta \mathbf{w} = -\frac{w_i}{(H^{-1})_{ii}}(H^{-1})\mathbf{e}_i$. Questo schema produce reti sparse di alta qualità ma richiede il calcolo dell'inversa Hessiana — fattibile solo con approssimazioni come la diagonale o le fattorizzazioni a basso rango. SparseGPT (Frantar e Alistarh 2023) adatta OBS ai LLM con la stessa strategia computazionale di GPTQ, raggiungendo il 50% di sparsità sui modelli LLaMA con meno del 1% di degradazione su benchmark linguistici standard.

Il **pruning iterativo** (Iterative Magnitude Pruning, IMP) alterna cicli di pruning e re-training: ad ogni iterazione si potano il $p\%$ dei pesi con magnitudine minore e si ri-addestra la rete sparsa fino a convergenza. Questo approccio produce reti più accurate rispetto al one-shot pruning perché il re-training permette ai pesi rimanenti di adattarsi alla nuova topologia sparsa. La scelta della sparsity target è guidata dal tradeoff accuratezza-latenza: sparsità $< 50\%$ produce benefici minimi su hardware denso, sparsità $> 90\%$ causa degradazione sostanziale, la finestra ottimale dipende fortemente dall'architettura e dal task specifico.

438.2 · Structured Pruning e Distanza dalla Sparse Inference

Il pruning non strutturato produce reti sparse difficili da accelerare su hardware reale: le GPU moderne sono ottimizzate per operazioni dense e il codice sparse ha overhead di indirizzamento che annulla i guadagni teorici sotto una densità di circa il 50–70%. Il **pruning strutturato** rimuove unità intere — neuroni, head di attenzione, layer — producendo una rete densa più piccola che beneficia direttamente dell'ottimizzazione hardware senza codice sparse. Il pruning

di attention head è supportato dall'analisi di Michel et al. (2019): molte head sono ridondanti — rimuovere fino al 40% delle head di BERT-Large causa degradazioni $<1\%$ su GLUE. Il **layer dropping** rimuove interi Transformer layer: Sajjad et al. (2020) mostrano che i layer intermedi di BERT contribuiscono meno dei layer iniziali e finali, e il dropping selettivo di layer intermedi preserva l'accuratezza meglio del dropping uniforme. Le **2:4 sparse tensors** introdotte da NVIDIA per Ampere e Ada Lovelace supportano un pattern specifico: in ogni blocco di 4 elementi, esattamente 2 sono non-zero, abilitando decompressione hardware e moltiplicazione sparsa a velocità quasi-dense. Il training con vincoli 2:4 (SparseGPT, Wanda) produce reti direttamente accelerabili su hardware NVIDIA senza overhead di runtime.

Knowledge Distillation: Trasferire la Conoscenza da Modello a Modello

Addestrare una rete piccola con le distribuzioni morbide di una grande — la dark knowledge di Hinton, la distillazione degli strati intermedi e le varianti moderni.

439.1 · La Dark Knowledge e la Distillazione Classica

La **knowledge distillation** (Hinton, Vinyals e Dean 2015) è la tecnica di compressione più efficace per trasferire le capacità di un modello grande (il *teacher*) in un modello più piccolo (lo *student*). L'intuizione chiave è che le **soft labels** — le distribuzioni di probabilità prodotte dal teacher — contengono più informazione delle hard labels (etichette one-hot): la distribuzione $[0.9, 0.05, 0.04, 0.01]$ comunica non solo che l'input è della classe 1, ma anche che è molto dissimile dalla classe 4 e abbastanza dissimile dalle classi 2 e 3. Questa *dark knowledge* sulla struttura delle relazioni tra classi è particolarmente utile per lo student. La loss di distillazione usa la cross-entropy con le soft labels del teacher alla temperatura T: $\mathcal{L}_{\text{KD}} = (1-\alpha)\mathcal{L}_{\text{CE}}(\mathbf{y}, \sigma(\mathbf{z}_S)) + \alpha T^2 \text{KL}(\sigma(\mathbf{z}_T/T) \| \sigma(\mathbf{z}_S/T))$ dove $\mathbf{z}_T$ e $\mathbf{z}_S$ sono i logit del teacher e dello student, e la temperatura $T > 1$ "ammorbidisce" le distribuzioni amplificando le probabilità delle classi meno probabili. Il fattore T^2 normalizza il gradiente della KL rispetto alla scala della temperatura. La distillazione è particolarmente efficace per la **compression semantica**: DistilBERT (Sanh et al. 2019) raggiunge il 97% delle prestazioni di BERT con il 40% dei parametri addestrando su soft labels e su un obiettivo di allineamento degli strati nascosti.

439.2 · Self-Distillation, Born-Again Networks e Distillazione Sequenziale

La distillazione non richiede necessariamente una gerarchia teacher-student: la **self-distillation** usa il modello stesso come teacher dopo il training e ri-addestra lo stesso modello sulle proprie

soft predictions. Furlanello et al. (2018) dimostrano con le **Born-Again Networks** che generazioni successive di self-distillazione migliorano monotonamente le prestazioni: la prima generazione impara dal modello base, la seconda dalla prima generazione, e così via — un fenomeno spiegato dalla riduzione della varianza delle predizioni grazie alle soft labels. La distillazione sequenziale si presta alla compressione progressiva: TinyBERT (Jiao et al., 2020) distilla BERT in fasi, prima distillando i layer Transformer (allineando le attention matrices e i hidden states) poi distillando il livello di task tramite fine-tuning sul dataset di interesse. La **data-free distillation** affronta il caso in cui i dati di training del teacher non sono disponibili per privacy o proprietà intellettuale: si generano dati sintetici massimizzando le attivazioni del teacher (DAFL, DeepInversion) o si usa un generatore avversariale addestrato a produrre esempi su cui teacher e student divergono massimalmente. Il **knowledge transfer cross-architettura** — distillare un Transformer in una RNN o CNN — è un caso estremo dove le differenze architetturali richiedono allineamento di feature a diversi livelli di astrazione, spesso usando proiezioni lineari apprese per allineare gli spazi di rappresentazione incommensurabili.

La **distillazione progressiva** (Salimans e Ho, 2022) è stata applicata ai diffusion model con grande successo: partendo da un modello che usa N passi, si addestra un modello student a riprodurre due passi del teacher in un singolo passo — dimezzando il numero di passi ad ogni round di distillazione. Dopo $\log_2 N$ round si ottiene un generatore che usa un singolo passo, mantenendo qualità superiore alla consistency distillation. Questa tecnica di distillazione iterativa è ora standard per la produzione di modelli di diffusione veloci in deployment commerciale, dove la latenza di generazione è critica per l'esperienza utente.

LoRA e Parameter-Efficient Fine-tuning

Adattare i LLM con una frazione dei parametri — l'ipotesi del rango basso, la derivazione di LoRA, AdaLoRA, QLoRA e le varianti del PEFT moderno.

440.1 · L'Ipotesi del Rango Basso e la Derivazione di LoRA

Il fine-tuning completo di un LLM da 70 miliardi di parametri richiede l'aggiornamento di tutti i 70 miliardi di pesi e la memorizzazione dei gradienti corrispondenti — proibitivo su hardware consumer. **LoRA (Low-Rank Adaptation)** (Hu, Shen, Wallis, Allen-Zhu, Li, Li, Wang e Chen 2022) si basa sull'osservazione empirica che gli aggiornamenti dei pesi durante il fine-tuning hanno *basso rango intrinseco*: le matrici di differenza $\Delta W = W_{\text{fine-tuned}} - W_{\text{pre-trained}}$ sono approssimabili con buona accuratezza da prodotti di matrici di rango basso $\Delta W \approx BA$

con $B \in \mathbb{R}^{d\times r}$, $A \in \mathbb{R}^{r\times k}$ e $r \ll \min(d,k)$. Durante il fine-tuning, i pesi originali W vengono congelati e si aggiornano solo le matrici A e B: $h = Wx + BAx = (W + BA)x$. A è inizializzata con distribuzione gaussiana e B con zero, garantendo che all'inizio del training $\Delta W = BA = 0$ e il modello si comporta identicamente al pre-trained. Il numero di parametri addestrabili è $r(d + k)$ invece di dk — una riduzione di un fattore $dk/(r(d + k)) \approx \min(d, k)/(2r)$. Con $r = 16$ su un modello LLaMA-7B, si addestrano solo 4 milioni di parametri invece di 7 miliardi — una riduzione del 99.94% nel numero di parametri, con la memoria GPU che scende da 14 GB a 1 GB per il modulo adattatore. **QLoRA** (Dettmers et al. 2023) combina LoRA con quantizzazione a 4 bit del modello base (usando NormalFloat4, un tipo di dato ottimizzato per pesi neurali), rendendo il fine-tuning di modelli da 65 miliardi di parametri fattibile su una singola GPU da 48 GB — un avanzamento democratizzante che ha permesso la creazione di decine di modelli specializzati open-source di alta qualità.

440.2 · Varianti PEFT: Prefix Tuning, Prompt Tuning e Adapter Layers

Prima di LoRA, il panorama del parameter-efficient fine-tuning era dominato da approcci diversi. Il **Prefix Tuning** (Li e Liang, 2021) aggiunge sequenze di vettori "prefisso" $P_K, P_V \in \mathbb{R}^{Limesd}$ alle chiavi e valori dell'attenzione in ogni layer, mantenendo il modello base congelato: l'attenzione diventa $\text{Attn}(Q, [P_K; K], [P_V; V])$ dove $[\cdot;\cdot]$ indica la concatenazione lungo la dimensione di sequenza. Solo il 0.1% dei parametri viene addestrato — le $2imesLimesd_{model}imesn_{layers}$ dimensioni del prefisso — con degradazione minima rispetto al full fine-tuning. Il **Prompt Tuning** (Lester et al., 2021) semplifica ulteriormente aggiungendo soft prompt tokens solo all'input del primo layer anziché a tutti i layer: con modelli da 11B parametri la differenza di qualità rispetto al full fine-tuning diventa trascurabile, mentre con modelli più piccoli il gap si amplia. Gli **Adapter layers** (Houlsby et al., 2019) inseriscono piccoli moduli bottleneck $\text{FFN}_{down} \circ \text{activation} \circ \text{FFN}_{up}$ dopo ogni sub-layer del Transformer: con dimensione bottleneck $r \ll d$, si aggiunge solo $2rd$ parametri per layer. **AdaLoRA** estende LoRA con allocazione adattiva del rank r per diversi layer e moduli, aumentando r dove il gradiente suggerisce maggiore importanza e riducendolo altrove — producendo migliore qualità a parità di parametri totali aggiornati.

Architetture Efficienti: da MobileNet a EfficientNet

Progettare reti per l'inferenza su dispositivi vincolati — depthwise separable convolutions, compound scaling e il ruolo del neural architecture search.

441.1 · Convoluzioni Separabili in Profondità

Le **MobileNet** (Howard et al. 2017) hanno introdotto le **depthwise separable convolutions** come alternativa efficiente alle convoluzioni standard. Una convoluzione standard $K \times K$ su un tensore $H \times W \times C_{\text{in}}$ verso C_{out} canali ha costo $O(H \cdot W \cdot C_{\text{in}} \cdot C_{\text{out}} \cdot K^2)$. La decomposizione in due operazioni sequenziali riduce drasticamente questo costo: la **depthwise convolution** applica un filtro $K \times K$ separatamente ad ogni canale di input (C_{in} filtri indipendenti, costo $O(H \cdot W \cdot C_{\text{in}} \cdot K^2)$), poi la **pointwise convolution** (1×1) mescola i canali (costo $O(H \cdot W \cdot C_{\text{in}} \cdot C_{\text{out}})$). Il rapporto di costo è $\frac{1}{C_{\text{out}}} + \frac{1}{K^2}$: per $K = 3$ e $C_{\text{out}} = 64$, la riduzione è di circa $8\times$. **EfficientNet** (Tan e Le 2019) introduce il **compound scaling**: invece di scalare solo la profondità (più layer), solo la larghezza (più canali) o solo la risoluzione (immagini più grandi), si scala proporzionalmente tutte e tre le dimensioni con un coefficiente ϕ: profondità $\propto 1.2^{\phi}$, larghezza $\propto 1.1^{\phi}$, risoluzione $\propto 1.15^{\phi}$, con il vincolo che il compute totale $\propto 2^{\phi}$. I coefficienti ottimali sono trovati tramite NAS su un budget limitato e poi usati per scalare il modello base a diverse dimensioni (EfficientNet-B0 a B7). Il risultato è che EfficientNet-B7 raggiunge il 84.4% di top-1 accuracy su ImageNet con 8 volte meno parametri e 6.1 volte meno FLOP di ResNet-152 — dimostrando che il compound scaling è la strategia di scalabilità ottimale quando si vuole massimizzare l'accuratezza per FLOP.

441.2 · Neural Architecture Search e Compound Scaling

La **Neural Architecture Search (NAS)** automatizza il processo di progettazione dell'architettura che EfficientNet ha ottimizzato manualmente per il compound scaling. **DARTS** (Liu et al., 2019) rilassa lo spazio discreto delle architetture in uno spazio continuo: ogni connessione della cella ricercata è una somma pesata di tutte le possibili operazioni candidate $\sum_o \frac{e^{\alpha_o}}{\sum_{o'} e^{\alpha_{o'}}} o(x)$, dove i pesi α_o sono parametri architetturali co-ottimizzati con i pesi della rete tramite gradient descent. Al termine della ricerca, l'architettura finale viene derivata selezionando l'operazione con peso massimo per ogni connessione. Il costo della ricerca DARTS è $O(1)$ rispetto alla ricerca per enumerazione, riducendo il tempo da migliaia a poche ore GPU. **MobileNetV3** combina NAS con Human-in-the-loop: architettura base cercata automaticamente, con ottimizzazioni manuali aggiuntive come hard-swish e squeeze-and-excitation sui layer critici identificati dal NAS. Le architetture **EfficientViT** e **FastViT** applicano il paradigma di efficienza ai Vision Transformer: rimpiazzano la self-attention $O(n^2)$ con attention lineare o window attention, mantenendo il campo recettivo globale con costo $O(n)$ tramite operazioni di token mixing depthwise-separabili. Il tradeoff accuratezza-latenza rimane il criterio dominante nel design di architetture per edge deployment, dove il numero di parametri è meno critico del profilo di latenza su hardware target specifico.

Overparameterization, Interpolazione e il Regime Benign

Perché le reti sovraparametrizzate generalizzano — l'interpolazione come condizione necessaria, benign overfitting e la distribuzione del rumore appresa.

442.1 · Il Regime di Interpolazione e il Minimo a Norma Minima

Un modello che **interpola** i dati di training è un modello con zero training error — non si generalizza ad alcuna tolleranza, memorizza ogni esempio. La statistica classica insegna che l'interpolazione porta al sovra-adattamento e è da evitare. Tuttavia, il deep learning moderno opera quasi sempre nel regime di interpolazione: reti con miliardi di parametri vengono addestrate fino a convergenza su milioni di esempi, raggiungendo zero training loss, e cionondimeno generalizzano eccellentemente. Belkin, Hsu, Ma e Mandal (2019) hanno formalizzato questo paradosso con il concetto di **benign overfitting**: esiste una classe di problemi statistici in cui i classificatori che interpolano i dati (incluso il rumore) generalizzano ottimalmente. Il meccanismo è la *decomposizione del segnale e del rumore* nell'interpolazione: in alta dimensione, il minimo a norma minima può adattarsi ai punti di rumore usando le direzioni irrilevanti dello spazio dei parametri (ortogonali al sottospazio del segnale) senza perturbarne le prestazioni sul test set. Formalmente, consideriamo la regressione ridge nel limite $\lambda \to 0$: $\hat{\theta} = (X^T X + \lambda I)^{-1} X^T y \to X^\dagger y$ (pseudo-inversa). Se $n \ll d$ (più parametri che dati), la soluzione è quella a norma minima e interpola esattamente. Il bias al test è determinato dalla componente del segnale nello spazio dei dati; la varianza è controllata dalla struttura spettrale di X. Per distribuzioni dei dati con decadimento spettrale rapido (comune nei dati reali), la varianza rimane piccola anche in assenza di regolarizzazione esplicita, producendo la doppia discesa (§358) e il benign overfitting.

442.2 · Benign Overfitting: Interpolazione senza Degrado Generalizzazione

Il fenomeno del **benign overfitting** (Bartlett et al., 2020) sfida la comprensione classica della curva bias-varianza: sotto certe condizioni, un modello che memorizza perfettamente i dati di training (interpolazione esatta, training loss = 0) può generalizzare ottimalmente ai dati di test. La condizione teorica richiede che il modello abbia struttura "spiked covariance": la maggior parte della varianza dei dati è concentrata in poche direzioni principali, e il modello usa queste direzioni per la predizione rilevante mentre le molteplici direzioni rumorose assorbono il rumore di training con contributo trascurabile alla varianza di predizione. Formalmente, per la regressione lineare con n campioni e $p \gg n$ feature, il minimo-norma interpolatore $\hat{\beta} = X^+(y)$

converge all'ottimo di Bayes quando il segnale è concentrato sui primi $k \ll n$ autovalori della matrice di covarianza e gli autovalori rimanenti decadono lentamente. Il framework del **lazy training** e del **Neural Tangent Kernel** (§445) fornisce un modello teorico per le reti neurali sovraparametrizzate: durante il training, i pesi si spostano di poco dalla loro inizializzazione, e la rete opera nel regime lineare attorno all'inizializzazione — permettendo un'analisi esatta della generalizzazione tramite teoria dei kernel. Le condizioni per il benign overfitting nelle reti neurali reali (non-lineari, oltre il regime NTK) restano oggetto di ricerca attiva.

La connessione tra overparameterizzazione e ottimizzazione è altrettanto importante: le reti sovraparametrizzate hanno loss landscape più "benigno" con meno saddle point e minimi locali. Choromanska et al. (2015) dimostrano sotto ipotesi di indipendenza che l'energia di tutti i minimi locali converge all'energia del minimo globale con l'aumentare della dimensione della rete — spiegando perché SGD trova soluzioni di buona qualità anche su loss non-convesse. Il **implicit bias del gradient descent** — la tendenza di GD con learning rate piccolo a convergere verso la soluzione di minima norma tra tutte quelle che interpolano i dati — è la chiave per comprendere perché le reti sovraparametrizzate generalizzano: tra i molti interpolatori esistenti, GD seleziona sistematicamente quelli con proprietà di regolarizzazione implicita favorevoli.

La Lottery Ticket Hypothesis: Trovare la Sotto-rete Vincente

Reti inizializzate casualmente contengono sotto-reti già efficaci — la prova dell'ipotesi originale, il winning ticket, e il collegamento con il pruning e il transfer.

443.1 · L'Ipotesi e la sua Prova

Frankle e Carlin (2019, ICLR Best Paper) hanno formulato la **Lottery Ticket Hypothesis (LTH)**: "Una rete neurale inizializzata casualmente contiene una sotto-rete — chiamata *winning ticket* — tale che, se addestrata in isolamento con la sua inizializzazione originale, può raggiungere prestazioni comparabili alla rete intera in un numero uguale o inferiore di passi." La prova sperimentale usa l'*iterative magnitude pruning with rewinding*: si addestra la rete completa $\theta^0 \to \theta^T$; si fa pruning dei pesi con magnitudine più bassa (rimuovendo una percentuale, tipicamente 20%, ad ogni iterazione); si *reimposta* i pesi rimanenti alla loro inizializzazione originale θ^0 (non θ^T); si riaddestra la sotto-rete risultante. Le sous-reti trovate con questo metodo — i "lottery tickets" — raggiungono tipicamente le prestazioni della rete originale con il 10–20% dei parametri su MNIST e CIFAR. L'ipotesi ha implicazioni profonde per la com-

prensione dell'ottimizzazione: suggerisce che il training non crea la struttura computazionale utile ma piuttosto *seleziona e raffina* strutture già latenti nell'inizializzazione casuale. Per le reti di grandi dimensioni (Transformer, ResNet-50), la LTH richiede "late rewinding" — il reset ai pesi dopo alcune iterazioni di training invece che all'inizializzazione — suggerendo che l'inizializzazione originale non è sufficiente e che una quantità di training iniziale è necessaria per stabilizzare la struttura.

443.2 · Winning Tickets, Linearmode Connectivity e Pruning at Initialization

La Lottery Ticket Hypothesis ha generato una linea di ricerca sulla struttura delle reti neurali appena prima e dopo il training. La **linear mode connectivity** (Frankle et al., 2020) osserva che un winning ticket trovato in un run di training è connesso linearmente alla rete completa finale nello spazio dei parametri: il percorso lineare $\theta = (1-t)\theta_{ticket} + t\theta_{full}$ ha loss pressoché costante per $t \in [0, 1]$, indicando che il training converge verso lo stesso bacino del winning ticket. Questo non vale per reti inizializzate casualmente — solo i ticket vincenti mostrano questa connettività. Il **pruning at initialization** (SNIP, GraSP, SynFlow) tenta di trovare la sotto-rete vincente prima di qualsiasi training, usando signal propagation o saliency scores calcolati al primo mini-batch o addirittura senza vedere i dati. SynFlow (Tanaka et al., 2020) usa uno score basato sul prodotto dei parametri lungo ogni percorso della rete, evitando layer collapse e producendo sotto-reti più bilanciate rispetto ai metodi basati su gradiente. Il problema centrale è che i metodi di pruning at initialization faticano a raggiungere le prestazioni del pruning post-training con re-training: il winning ticket sembra emergere durante il processo di training stesso, non essere predeterminato dall'inizializzazione — in contrasto con la versione forte della Lottery Ticket Hypothesis originale.

La **sparse training from scratch** cerca di trovare la sotto-rete vincente e addestrarla dall'inizio senza la fase di training della rete densa seguita da pruning. SET (Sparse Evolutionary Training, Mocanu et al., 2018) e RigL (Evci et al., 2020) mantengono una rete sparsa durante il training aggiornando dinamicamente la topologia: ad ogni intervallo si rimuovono i pesi con magnitudine minore e si aggiungono nuovi pesi nelle posizioni con gradiente maggiore — esplorando lo spazio delle topologie sparse senza mai materializzare la rete densa. RigL raggiunge accuratezza paragonabile al pruning post-training su ResNet-50 con sparsità del 90%, richiedendo solo $1.5 \mathit{imes}$ il training della rete densa anziché i $3 \mathit{imes}$ tipici del ciclo train-prune-retrain.

Grokking e le Transizioni di Fase nell'Apprendimento

La generalizzazione ritardata e improvvisa — il fenomeno del grokking su aritmetica modulare, la spiegazione tramite la complessità effettiva e le transizioni di fase nei LLM.

444.1 · Il Fenomeno del Grokking

Power, Lightman, Khan, Buchanan e Bhatt (2022, OpenAI) hanno osservato un fenomeno sorprendente nell'addestramento di transformer su task di aritmetica modulare semplice (somma di interi modulo un primo p): dopo aver raggiunto zero training error molto presto nel training, la validation accuracy rimane vicina al caso (generalizzazione nulla) per un numero molto grande di passi aggiuntivi, poi improvvisamente balza alla precisione quasi-perfetta. Questo fenomeno — chiamato **grokking** — contraddice l'aspettativa che zero training error implichi automaticamente buona generalizzazione, e mostra che la generalizzazione può essere "ritardata" di ordini di grandezza rispetto alla memorizzazione. La spiegazione di Nanda, Chan, Lieberum, Smith e Steinhardt (2023) è meccanicistica: il modello impara prima una soluzione di memorizzazione (lookup table efficiente) e poi, durante il training prolungato, viene "incoraggiato" dalla regolarizzazione a trovare un algoritmo generalizzante più semplice (basato sulle armoniche di Fourier del gruppo ciclico $\mathbb{Z}_p$). La transizione avviene quando l'algoritmo di Fourier diventa più efficiente in termini di norma dei pesi rispetto alla lookup table. Questo suggerisce che la **complessità effettiva** (misurata dalla norma dei pesi) è il vero indicatore della generalizzazione, e che un training sufficientemente lungo con regolarizzazione implicita da weight decay porta il modello verso soluzioni più semplici anche dopo aver già memorizzato i dati.

444.2 · Meccanismi del Grokking: Generalizzazione Ritardata e Fasi di Apprendimento

L'analisi meccanicistica del grokking (Nanda et al., 2023) su addizione modulare rivela che il modello attraversa fasi di apprendimento qualitativamente distinte. Nella **fase di memorizzazione**, il modello usa un circuito complesso e ridondante per memorizzare i risultati; nella **fase di generalizzazione**, il modello sviluppa un circuito più semplice basato su trasformate di Fourier — la somma $a + b \pmod{p}$ viene calcolata codificando a e b come frequenze sinusoidali e usando il teorema dell'addizione trigonometrica. La transizione tra le due fasi avviene quando il circuito generalizzante diventa sufficiente da solo, e la weight decay progressivamente penalizza il circuito di memorizzazione ridondante fino alla sua eliminazione. Il grokking è quindi una manifestazione di **competizione tra circuiti**: il circuito generalizzante ha norma parametrica inferiore e viene favorito dalla regularizzazione, ma richiede più training

per emergere. Le **transizioni di fase nell'apprendimento** osservate in LLM — capacità emergenti che appaiono improvvisamente oltre una certa scala (BIG-Bench, Wei et al., 2022) — potrebbero essere manifestazioni dello stesso fenomeno a scala molto maggiore: una capacità "appare" quando il circuito generalizzante corrispondente supera la soglia di dominanza sul circuito di memorizzazione in tutti i modelli abbastanza grandi da avere sufficiente capacità per svilupparlo.

Il **double descent nel grokking** collega il fenomeno a dinamiche di ottimizzazione più generali: la curva di training loss mostra prima un rapido decremento (memorizzazione), poi una fase di stagnazione (competizione tra circuiti), poi un secondo decremento più lento (emergenza del circuito generalizzante). Questo pattern si osserva sull'addizione modulare, sulla composizione di permutazioni e su task algoritmici più complessi. La **weight decay come catalizzatore** è cruciale: senza regularizzazione il grokking non avviene perché il circuito di memorizzazione, avendo norma parametrica maggiore, viene penalizzato più fortemente dal weight decay — creando una pressione evolutiva verso il circuito più efficiente in termini di norma. Questo fornisce una spiegazione meccanicistica del perché la regularizzazione migliora la generalizzazione oltre l'overfitting classico: non solo evita l'overfitting ma attivamente seleziona soluzioni algoritmicamente più compatte.

La Geometria delle Loss Landscape

Minimi, selle, barriere di loss e le proprietà topologiche della superficie di ottimizzazione — il linear mode connectivity, la loss barrier e il mode averaging.

445.1 · Minimi Piatti vs. Minimi Appuntiti

La **loss landscape** di una rete neurale è la funzione $\mathcal{L} : \mathbb{R}^p \to \mathbb{R}$ dove p è il numero di parametri. Nonostante la sua altissima dimensionalità rendesse la visualizzazione diretta impossibile, Goodfellow, Vinyals e Saxe (2015) e poi Li et al. (2018) hanno sviluppato tecniche di *proiezione monodimensionale e bidimensionale* che visualizzano la loss lungo direzioni casuali (o direzioni significative) nello spazio dei parametri. Le reti più profonde e overparametrizzate mostrano loss landscape con molti **minimi piatti** — regioni ampie nello spazio dei parametri dove la loss è bassa — in contrasto con i minimi "stretti" di modelli più piccoli. La metrica standard per la "piattezza" è la sharpness $\lambda_{\max}(H_{\mathcal{L}}(\theta))$, il massimo autovalore della Hessiana della loss. L'ipotesi *flat minima = better generalization* (Hochreiter e Schmidhuber 1997) afferma che i minimi piatti generalizzano meglio perché la distribuzione di test leggermente diversa

da quella di training porta a una loss simile. Kaur et al. (2023) hanno mostrato che la relazione non è così semplice: la piattezza dipende dalla parametrizzazione usata e non è un invariante sotto le reparametrizzazioni equivalenti del modello. **SAM (Sharpness-Aware Minimization)** (Foret et al. 2021) opera su questa intuizione: modifica l'obiettivo di training per minimizzare la worst-case loss in un intorno ρ del punto corrente nello spazio dei parametri, $\min_\theta \max_{\|\epsilon\|\leq\rho} \mathcal{L}(\theta+\epsilon)$, producendo soluzioni più piatte e con migliore generalizzazione su benchmark di visione a parità di architettura.

445.2 · Linear Mode Connectivity e il Mode Averaging

Il **Linear Mode Connectivity (LMC)** (Nagarajan e Kolter 2019, Frankle et al. 2020) studia se due soluzioni θ_A e θ_B trovate da SGD sono connesse da un percorso lineare a bassa loss: $\mathcal{L}((1-\lambda)\theta_A + \lambda\theta_B) \lesssim \max(\mathcal{L}(\theta_A), \mathcal{L}(\theta_B))$ per ogni $\lambda \in [0,1]$. Se la perdita sul percorso lineare è bassa, i due minimi sono nella stessa "basin of attraction e si può combinarli per migliorare le prestazioni. Notevolmente, due repliche dello stesso modello addestrate con seed diversi *non* sono generalmente linearly connected — le permutazioni dei neuroni nascosti rompono la connettività. L'algoritmo di **permutation matching** (Ainsworth, Hayase e Srinivasa 2022, "Git Re-Basin") trova la permutazione dei pesi che rende due reti linearly connected: riallineando i neuroni corrispondenti, la media dei pesi produce una terza rete con prestazioni migliori di entrambe le originali. Questo ha applicazioni dirette al **Model Merging**: invece di addestrare modelli separati per task diversi e fare fine-tuning su tutti, si addestrano modelli separati e si fa la media dei loro pesi allineati, producendo un unico modello multi-task senza costo di training aggiuntivo — una tecnica usata in produzione per sistemi come Mistral e Yi.

445.4 · Sharpness-Aware Minimization e la Geometria della Generalizzazione

La connessione tra geometria del loss landscape e generalizzazione è stata quantificata da misure di **sharpness**: minimi piatti (flat minima) generalizzano meglio di minimi stretti (sharp minima) perché sono più robusti a piccole perturbazioni dei parametri e alle differenze tra distribuzione di training e test. Hochreiter e Schmidhuber (1997) proposero per primi questa intuizione; Keskar et al. (2017) la verificarono empiricamente mostrando che il batch size grande produce sharp minima con peggiore generalizzazione. **Sharpness-Aware Minimization (SAM)** (Foret et al., 2021) ottimizza esplicitamente per flat minima: invece di minimizzare $\mathcal{L}(\theta)$, minimizza $\max_{\|\epsilon\|\leq\rho} \mathcal{L}(\theta+\epsilon)$ — la worst-case loss in un intorno di raggio ρ attorno ai parametri correnti. Il gradiente di questo obiettivo richiede due forward/backward pass: prima si trova la perturbazione $\hat{\epsilon}$ che massimizza la loss (un passo di gradient ascent), poi si aggiornano i parametri usando il gradiente calcolato in $\theta+\hat{\epsilon}$. SAM produce consistente miglio-

ramento di accuratezza (1–2%) e robustezza alla distributional shift su ImageNet, CIFAR e NLP benchmark, a un costo computazionale doppio. Varianti efficienti come ASAM (adaptive SAM) e GSAM (guided SAM) riducono il costo o migliorano la qualità dell'approssimazione della sharpness in spazi parametrici non isotropi.

Memorizzazione vs. Generalizzazione: la Dicotomia Fondamentale

Quando le reti memorizzano e quando generalizzano — il training su etichette corrotte, la distanza di Kolmogorov e il ruolo della struttura dei dati.

446.1 · La Linea tra Memorizzare e Generalizzare

L'esperimento di **Zhang et al. (2017)** "Understanding Deep Learning Requires Rethinking Generalization" ha dimostrato sperimentalmente che le reti neurali possono memorizzare qualunque dataset di training: sostituendo le etichette di CIFAR-10 con etichette completamente casuali, la rete raggiunge zero training error (memorizzazione perfetta) in circa lo stesso numero di passi necessari con etichette reali — solo modestamente più lento. La rete ha quindi la capacità di memorizzare qualunque mappa dato-etichetta per i 50.000 esempi di training. Eppure con etichette vere generalizza; con etichette casuali non generalizza affatto. La differenza non è nella rete — è nei *dati*. Questo suggerisce che la generalizzazione non è una proprietà del modello isolato ma dell'interazione tra modello, algoritmo di ottimizzazione, e struttura dei dati. La **Kolmogorov Complexity perspective** (Schmidhuber 1997, Solomonoff 1964) fornisce un framework: i dati con struttura regolare (immagini naturali, testo, musica) hanno bassa complessità di Kolmogorov — possono essere compressi in programmi corti. Una rete neurale che apprende da questi dati impara implicitamente un programma di compressione corto, che generalizza perché riflette la struttura sottostante. I dati con etichette casuali hanno alta complessità di Kolmogorov — richiedono programmi lunghi (lookup table) per essere descritti — e il "programma" imparato dalla rete non comprime, non generalizza. Questa prospettiva unifica la teoria della generalizzazione (PAC learning, Rademacher complexity) con la teoria della compressione (MDL, §357) e suggerisce che il problema della generalizzazione è fondamentalmente un problema di stima della complessità strutturale dei dati.

446.2 · Memorizzazione come Risorsa: Long-tail e Retrieval-Augmented Generation

La dicotomia memorizzazione-generalizzazione è stata rivalutata alla luce del **teorema di necessità della long-tail memorization** (Feldman, 2020): se la distribuzione dei dati ha una

coda lunga, un classificatore ottimale deve necessariamente memorizzare gli esempi rari — non può generalizzare da esempi frequenti a istanze di distribuzioni long-tail. Formalmente, se la distribuzione empirica assegna probabilità p_i all'istanza i con p_i piccolo, l'errore di un classificatore che non memorizza i è minorizzato da $\Omega(p_i)$ — giustificando la memorizzazione come strategia ottimale per le istanze rare. Per i LLM questo implica che la memorizzazione di fatti rari è funzionalmente necessaria. La **memorizzazione selettiva** distingue memorizzazione verbatim — riproduzione di sequenze di training, principalmente per testi ad alta frequenza, codice e formule — da memorizzazione semantica, che codifica il significato senza riproduzione letterale. Il **Retrieval-Augmented Generation (RAG)** esternalizza la memorizzazione fattuale verso un database ricercabile: il modello necessita solo di capacità di ragionamento e follow-the-context, lasciando i fatti aggiornabili, citabili e verificabili nella knowledge base esterna. Questo crea un'architettura ibrida dove parametri e memoria esterna svolgono ruoli complementari — i parametri codificano procedure e pattern generali, la memoria esterna fornisce fatti specifici e aggiornabili — avvicinando il design dei sistemi AI alla distinzione umana tra memoria procedurale e memoria dichiarativa episodica.

Modelli Vision-Language: CLIP, Flamingo e GPT-4V

Apprendimento congiunto di visione e linguaggio — contrastive pre-training su scala web, architetture cross-modale e capacità emergenti nei modelli vision-language di larga scala.

447.1 · CLIP: Contrastive Language-Image Pre-training

Il **CLIP** (Contrastive Language-Image Pre-training, Radford et al., 2021) rivoluziona il pre-training visivo addestrando simultaneamente un image encoder e un text encoder a massimizzare la similarità coseno tra le rappresentazioni di coppie immagine-testo corrette e minimizzarla per le coppie scorrette. Il dataset di training — 400 milioni di coppie immagine-alt$_t$*extraccoltedalweb*−−−*ordinidigrandezzapigrandedi**ImageNet**ecopreunadistribuzionesemanticavas*imes N*dovesimassimizzaladiagonale* :L = -1$\frac{}{N \sum_i \log \frac{\exp(\text{sim}(I_i, T_i)/au)}{\sum_j \exp(\text{sim}(I_i, T_j)/au)}}$ dove au è una temperatura appresa. La zero-shot classification con CLIP converte ogni classe in un template testuale ("a photo of a class"), calcola gli embedding di testo per tutte le classi, poi assegna l'immagine alla classe con embedding più simile — senza esempi etichettati e senza fine-tuning. Su ImageNet, CLIP zero-shot raggiunge il 76.2% di accuratezza, paragonabile a ResNet-50 supervisionato, generalizzando eccezionalmente su distribuzioni di test shifted (ImageNet-V2, -R, -Sketch) dove i modelli supervisionati degradano significativamente.

L'embedding space di CLIP è notevolmente strutturato: immagini e descrizioni testuali della stessa scena si trovano vicine, permettendo operazioni aritmetiche cross-modali. Il **DALL-E 2** (Ramesh et al., 2022) sfrutta questa struttura usando CLIP image embedding come condizionamento per un diffusion prior — generando immagini fedeli a descrizioni testuali attraverso lo spazio latente CLIP condiviso. **SigLIP** (Zhai et al., 2023) sostituisce l'obiettivo softmax normalizzato di CLIP con una loss sigmoid per coppia, eliminando la dipendenza dalla dimensione del batch e permettendo training più stabile su batch piccoli o con dati di qualità variabile. **DFN** (Data Filtering Networks) dimostra che la qualità del dato supera la quantità: filtrare 5 miliardi di coppie con un modello di filtraggio produce feature migliori di usarne 12 miliardi non filtrate.

447.2 · Flamingo e GPT-4V: Few-Shot Multimodale

Flamingo (Alayrac et al., 2022) introduce l'architettura per i modelli vision-language few-shot: un LM pre-addestrato viene "congelato" e condizionato su immagini tramite **Perceiver Resampler** — un modulo cross-attention che proietta un numero variabile di feature visive in un numero fisso di token visivi ($N_{vis} = 64$) che il LM può processare. Layer **Gated Cross-Attention** inseriti tra i layer del LM permettono all'informazione visiva di fluire nel flusso testuale: $h \leftarrow h + anh(\alpha) \cdot \text{CrossAttn}(h, x_{vis})$ dove α è inizializzato a zero per preservare le capacità originali del LM. Flamingo con 80B parametri raggiunge prestazioni few-shot (con 4 esempi di contesto) superiori ai modelli fine-tuned state-of-the-art su VQA, captioning e OCR. **GPT-4V** (OpenAI, 2023) e **Gemini** (Google, 2023) estendono questa architettura a scale massicce con dati di training multimodali curati, mostrando capacità emergenti come ragionamento visivo su diagrammi complessi, comprensione di codice in screenshot e analisi medica di immagini cliniche — capacità non presenti nei modelli più piccoli anche con fine-tuning dedicato.

State Space Models: S4, Mamba e il Regime Sub-Quadratico

Modelli di sequenza oltre il Transformer — la formulazione SSM, la convoluzione diagonale di Hippo-LegS, la selective state space di Mamba e la loro posizione nel tradeoff efficienza-espressività.

448.1 · Structured State Space Models (S4)

I **Structured State Space Models (S4)** (Gu et al., 2021) offrono un'alternativa ai Transformer per la modellazione di sequenze lunghe, basata sulla teoria dei sistemi lineari tempo-invarianti. Il modello S4 evolve uno stato latente $\mathbf{h}(t) \in \mathbb{R}^N$ secondo l'equazione differenziale $\dot{\mathbf{h}}(t) = \mathbf{A}\mathbf{h}(t) + \mathbf{B}x(t)$, $y(t) = \mathbf{C}\mathbf{h}(t) + Dx(t)$, dove $\mathbf{A} \in \mathbb{R}^{NimesN}$ è la matrice di transizione. Discretizzando con il metodo di Euler o bilineare si ottiene una ricorrenza $\mathbf{h}_t = \bar{\mathbf{A}}\mathbf{h}_{t-1} + \bar{\mathbf{B}}x_t$. Il problema chiave è che il calcolo ricorrente è sequenziale — $O(TN)$ con T passi e stato di dimensione N. La chiave di S4 è che per matrici $\mathbf{A}$ di struttura speciale (Normal Plus Low-Rank, inizializzata secondo la teoria **HiPPO** che approssima ottimalmente polinomi di Legendre), la risposta impulsiva del sistema è calcolabile in forma chiusa come convoluzione con kernel $\bar{\mathbf{K}} = (\mathbf{C}\bar{\mathbf{B}}, \mathbf{C}\bar{\mathbf{A}}\bar{\mathbf{B}}, \ldots, \mathbf{C}\bar{\mathbf{A}}^{T-1}\bar{\mathbf{B}})$, permettendo il calcolo parallelizzato con FFT in $O(T \log T)$. S4 supera i Transformer su task con dipendenze ultra-long-range (PathX con sequenze da 16k token) dove l'attenzione degrada.

448.2 · Mamba: Selective State Spaces

Mamba (Gu e Dao, 2023) risolve la principale limitazione di S4: la matrice $\mathbf{A}$ è fissa e le proiezioni $\mathbf{B}, \mathbf{C}$ sono costanti — il modello non può selezionare quali parti dell'input ricordare in modo content-dependent. Mamba introduce la **selective scan**: i parametri $\mathbf{B}_t, \mathbf{C}_t, \Delta_t$ diventano funzioni dell'input corrente x_t, calcolate tramite proiezioni lineari leggere. Questo permette al modello di "aprire" o "chiudere" il gate dello stato a seconda del contenuto, emulando il meccanismo di selezione dell'attenzione a costo lineare $O(T)$ in memoria e $O(T)$ in tempo — contro $O(T^2)$ dell'attenzione standard. Il costo di questo approccio è che la scan selettiva non è parallelizzabile con FFT (i parametri dipendono dall'input): Mamba risolve questo con un **hardware-aware parallel scan** che sfrutta le operazioni associative per parallelizzare la ricorrenza su GPU. L'architettura Mamba-2 (Dao e Gu, 2024) unifica SSM e attenzione come casi speciali di **structured matrix multiplication**, mostrando che Mamba è equivalente a un'attenzione lineare con gating strutturato. Modelli basati su Mamba come Jamba (ibrido Mamba+Transformer) e Zamba raggiungono qualità paragonabile ai Transformer su benchmark linguistici standard con throughput di inferenza 3-5× superiore per contesti lunghi.

Retrieval-Augmented Generation: Architettura e Ottimizzazione

RAG come paradigma di knowledge augmentation — retriever denso, re-ranking, chunk strategy, e i metodi di fine-tuning end-to-end per ottimizzare retriever e generator congiuntamente.

449.1 · Dense Passage Retrieval e il Pipeline RAG

Il **Retrieval-Augmented Generation (RAG)** (Lewis et al., 2020) divide il problema della question answering knowledge-intensive in due sotto-problemi: recupero (*retrieve*) dei documenti rilevanti da una knowledge base e generazione (*generate*) della risposta condizionata sui documenti recuperati. Il **Dense Passage Retrieval (DPR)** codifica query e documenti in vettori densi tramite due encoder BERT separati e usa similarità del prodotto scalare per il recupero — sostituendo il BM25 sparso con ricerca vettoriale approssimata (FAISS, HNSW) scalabile a milioni di documenti con latenza di pochi millisecondi. Il pipeline standard RAG processa ogni query in tre fasi: (1) encoding della query con il retriever, (2) ricerca dei top-k chunk più simili nell'indice vettoriale, (3) concatenazione dei chunk come contesto nel prompt del LM. La qualità del RAG dipende fortemente dalla strategia di **chunking**: chunk troppo piccoli perdono contesto, chunk troppo grandi saturano la finestra del LM e diluiscono la rilevanza. Chunk ricorsivi gerarchici (512 token con overlap del 20%), sentence-level splitting con aggregazione semantica (SemanticChunker) e document-level summary prodotti da un LM sono strategie comuni con trade-off diversi.

449.2 · Advanced RAG: Re-ranking, HyDE e RAPTOR

Il retrieval a singolo stadio soffre di **vocabulary mismatch** tra query e documenti e di rumore nei chunk recuperati. Il **re-ranking a due stadi** usa un cross-encoder (un BERT che vede query e documento concatenati) per riordinare i top-k risultati del retriever biencoder: il cross-encoder è più preciso perché può modellare l'interazione diretta query-documento, ma troppo lento per scan sull'intero corpus. **HyDE** (Hypothetical Document Embeddings, Gao et al., 2022) affronta il mismatch generando un documento ipotetico che risponderebbe alla query usando il LM, poi recuperando i documenti reali più simili all'embedding del documento ipotetico — sfruttando la distribuzione dello spazio documentale invece di quella delle query. **RAPTOR** (Sarthi et al., 2024) costruisce un albero di riassunti gerarchici sul corpus: le foglie sono i chunk originali, i nodi interni sono riassunti progressivamente più astratti generati da un LM, e il retrieval avviene a tutti i livelli dell'albero simultaneamente — catturando sia dettagli locali sia struttura tematica globale. **FLARE** e **Self-RAG** introducono il retrieval adattivo: il LM decide quando recuperare (non ogni query lo richiede) tramite token speciali o classificatori interni, riducendo la latenza media e migliorando la coerenza delle risposte su query che non richiedono conoscenza esterna.

Reasoning e Planning nei LLM: Chain-of-Thought Avanzato e Tree Search

Dalle catene di ragionamento lineari alla ricerca ad albero — Tree of Thoughts, MCTS applicato ai LLM, process reward model e il paradigma del test-time compute scaling.

450.1 · Tree of Thoughts e Ricerca Euristica

Il **Chain-of-Thought** standard (§412) genera un singolo percorso di ragionamento lineare — un'euristica greedy che non esplora percorsi alternativi quando quello corrente è subottimale. Il **Tree of Thoughts (ToT)** (Yao et al., 2023) generalizza CoT a una ricerca ad albero: ad ogni passo il LM genera k "pensieri" candidati (nodi dell'albero), valuta la loro promettenza tramite un'altra chiamata al LM come euristica, e procede ricorsivamente con BFS o DFS. La struttura ad albero permette **backtracking**: se un ramo si rivela non produttivo, l'esplorazione torna al nodo padre e prova un'altra direzione — impossibile con CoT lineare. Su Game of 24 (formare 24 con 4 numeri e operazioni aritmetiche), ToT risolve il 74% dei problemi contro il 4% di CoT standard. Il **Graph of Thoughts** (Besta et al., 2023) estende ulteriormente a grafi generali: i pensieri possono essere aggregati (unendo idee da rami diversi) e raffinati (applicando trasformazioni a un pensiero esistente), modellando processi cognitivi più ricchi della semplice ricerca ad albero.

450.2 · Process Reward Models e Test-Time Compute Scaling

I **Process Reward Models (PRM)** (Lightman et al., 2023) addestrano un modello separato a valutare la correttezza di ogni singolo passo di ragionamento, non solo della risposta finale — superando i limitation degli Outcome Reward Models (ORM) che possono premiare ragionamenti errati che casualmente producono la risposta giusta. Il PRM viene addestrato su annotazioni umane di passi corretti/errati in soluzioni di problemi di matematica, producendo un segnale di ricompensa denso usabile per ricerca e per RLHF passo-passo. Il paradigma del **test-time compute scaling** (Snell et al., 2024) mostra che allocare più compute durante l'inferenza — generando e selezionando tra molte soluzioni candidate tramite un PRM — migliora le prestazioni in modo predicibile seguendo leggi di scala analoghe a quelle del pre-training. AlphaCode 2, o1 di OpenAI e DeepSeek-R1 sfruttano questa idea: il modello genera molte soluzioni candidate, il PRM seleziona la migliore, raggiungendo prestazioni su matematica olimpionica che nessun modello standard raggiunge indipendentemente dalla scala. Il tradeoff fondamentale è latenza vs qualità: il test-time scaling aumenta linearmente la latenza con il numero di campioni, rendendo necessario un sistema di routing che alloca più compute alle query difficili e meno a quelle facili.

Video Generation: Architetture e Modelli Fondazionali per il Video

Estendere la diffusione a sequenze temporali — 3D U-Net, Video Transformer, modellazione della consistenza temporale e il problema della scala computazionale per la generazione video di alta qualità.

451.1 · Video Diffusion Models e Attenzione Spazio-Temporale

La generazione video affronta sfide aggiuntive rispetto alla generazione di immagini: la **consistenza temporale** (oggetti e illuminazione devono rimanere coerenti tra frame), il **motion realism** (le traiettorie devono rispettare la fisica) e la **scala computazionale** (un video di 5 secondi a 24fps è 120 immagini, $120\times$ il costo di un'immagine). I **Video Diffusion Models** (Ho et al., 2022) estendono la U-Net 2D a 3D aggiungendo convoluzione temporale e attenzione temporale tra i layer spaziali esistenti: ogni residual block processa prima la dimensione spaziale con convoluzioni 2D poi la dimensione temporale con convoluzione 1D, permettendo l'uso di pesi pre-addestrati sull'immagine con fine-tuning solo sui nuovi layer temporali. Il **factorized space-time attention** separa l'attenzione spaziale (su ogni singolo frame) dall'attenzione temporale (tra i frame alla stessa posizione spaziale), riducendo il costo da $O((HWT)^2)$ a $O((HW)^2 + T^2)$ — fondamentale per mantenere la computazione trattabile. Make-A-Video (Singer et al., 2022) e Imagen Video (Ho et al., 2022) usano architetture a cascata: un modello genera video a bassa risoluzione e basso framerate, poi modelli di super-resolution temporale e spaziale refinano progressivamente la qualità.

451.2 · Sora e i Video World Model

Sora (OpenAI, 2024) ridefinisce lo stato dell'arte con un'architettura basata su **Diffusion Transformer (DiT)** applicato a patch video spazio-temporali. I video vengono prima compressi in uno spazio latente 3D tramite un Video VAE, poi decomposti in patch spazio-temporali processate da un DiT. L'uso di patch variabili per durata e risoluzione permette di addestrare su video eterogenei (durata 1–60 secondi, risoluzione SD-4K) in un unico modello senza ridimensionamento forzato — un progresso fondamentale rispetto alle architetture che richiedevano preprocessing uniforme. Sora dimostra capacità di **world modeling emergente**: la coerenza 3D degli oggetti nel corso del video, la persistenza di oggetti occludendosi e riapparendo, e la fisica approssimativa del movimento suggeriscono che il modello apprende una rappresentazione interna della struttura del mondo tridimensionale e delle sue dinamiche — non solo pattern superficiali di pixel. **CogVideoX** (Yang et al., 2024) e **Wan** (Alibaba, 2025) sono

modelli open-source che raggiungono qualità comparabile usando architetture 3D-DiT con attenzione full-sequence su video compressi, rendendo la ricerca sulla generazione video accessibile senza infrastruttura proprietaria.

LLM Agent: Tool Use, Planning e Sistemi Multi-Agente

I LLM come agenti autonomi — architetture ReAct e Reflexion, tool calling strutturato, orchestrazione multi-agente e i benchmark per la valutazione delle capacità agentiche.

452.1 · ReAct, Tool Calling e Architetture Agentiche

Il paradigma **ReAct** (Yao et al., 2023) intercala ragionamento (Thought) e azione (Action) in un loop iterativo: il LM genera un pensiero che analizza la situazione corrente, poi emette un'azione (ricerca web, esecuzione codice, chiamata API), osserva il risultato (Observation), e itera fino al completamento del task. Questo schema pensiero-azione-osservazione trasforma il LM da generatore di testo in **agente che pianifica e interagisce con strumenti**. Il **Function Calling** strutturato (OpenAI, Anthropic) formalizza il tool use: ogni strumento è descritto con uno schema JSON (nome, parametri, tipi), il LM genera chiamate di funzione strutturate che vengono parsate e eseguite deterministicamente, e i risultati vengono reinseriti nel contesto. Questo elimina il parsing fragile del testo libero sostituendolo con interfacce strutturate tipizzate. **Reflexion** (Shinn et al., 2023) aggiunge auto-riflessione: dopo ogni episodio fallito, il LM genera una critica verbale del proprio comportamento che viene memorizzata ed usata come contesto negli episodi successivi — una forma di apprendimento da errori senza gradient descent, usando la memoria episodica del contesto come surrogate per i parametri.

452.2 · Sistemi Multi-Agente e Orchestrazione

I **sistemi multi-agente LLM** distribuiscono task complessi tra agenti specializzati con ruoli distinti. Il framework **AutoGen** (Wu et al., 2023) coordina conversazioni strutturate tra agenti: un agente orchestratore assegna sotto-task agli agenti worker, raccoglie i risultati e verifica la coerenza complessiva. La **society of mind** (Du et al., 2023) usa debate multi-agente: più LM generano risposte indipendenti, poi si esaminano reciprocamente in più round di discussione, convergendo verso risposte più accurate di quelle di un singolo agente. I benchmark per agenti includono WebArena (navigazione web autonoma), SWE-bench (risoluzione di bug reali su GitHub) e GAIA (task di ragionamento multi-step con strumenti): su SWE-bench, i migliori sistemi agentici risolvono il 30–50% dei bug, contro il 2–5% dei LM non agentici — misurando

il guadagno reale del paradigma agentico su task ingegneristici. I limiti attuali sono il **context length e la reliabilità**: in task lunghi la qualità del ragionamento degrada con la lunghezza del contesto, e gli errori di pianificazione si propagano irrecuperabilmente senza meccanismi di backtracking robusti.

World Models Neurali: Dreamer, RSSM e Simulazione Interna

Il paradigma del world model per il reinforcement learning — rappresentazione latente del mondo, pianificazione nel latent space, e i modelli generativi come simulatori interni per il training degli agenti.

453.1 · RSSM e Dreamer: Pianificazione nel Latente

Un **world model** è un modello generativo che apprende a predire le conseguenze delle azioni nel proprio spazio di rappresentazione interna, permettendo all'agente di pianificare *immaginando* le traiettorie future senza interagire con l'ambiente reale. Il **Recurrent State Space Model (RSSM)** di Hafner et al. (2019) combina uno stato deterministico h_t (aggiornato da un GRU) con uno stato stocastico z_t (campionato da un encoder variadionale): $h_t = f(h_{t-1}, z_{t-1}, a_{t-1})$, $z_t \sim q_\phi(z_t \mid h_t, x_t)$. La separazione tra componente deterministica (memoria a lungo termine) e stocastica (incertezza a breve termine) permette di generare traiettorie coerenti su orizzonti lunghi. **Dreamer** (Hafner et al., 2020) addestra simultaneamente il world model (ricostruzione delle osservazioni), l'actor (policy) e il critic (value function) interamente nelle traiettorie immaginarie generate dal world model — senza gradient attraverso l'ambiente reale. Le policy imparate nel latente si trasferiscono all'ambiente reale perché il world model impara una rappresentazione fedele della struttura dell'ambiente. **DreamerV3** (Hafner et al., 2023) estende questo framework a un range di task eterogeneo (Atari, robotica, Minecraft) con un singolo set di iperparametri, il primo agente generalista basato su world model a superare prestazioni umane su molti benchmark senza hyperparameter tuning specifico per task.

453.2 · JEPA e Rappresentazioni Predittive Latenti

La **Joint Embedding Predictive Architecture (JEPA)** (LeCun, 2022) propone un paradigma alternativo ai generative model tradizionali per costruire world model: invece di predire pixel futuri nello spazio di input (costoso e spesso inutile), il modello predice la **rappresentazione** futura in uno spazio di embedding astratto. Data un'osservazione x e un'azione a, il JEPA predice $\hat{s}_{t+1} = g_h eta(s_t, a_t)$ dove $s_t = f_\phi(x_t)$ è l'embedding dell'osservazione corrente — evi-

tando di modellare dettagli irrilevanti come texture e rumore. **V-JEPA** (Assran et al., 2023) applica questo principio al video: una rete di encoding processa i frame visibili, un predictor predice le rappresentazioni dei frame mascherati nello spazio latente. La valutazione avviene su probe di classification lineari — se le rappresentazioni sono buone, un classificatore lineare deve performare bene. I-JEPA ottiene SOTA su ImageNet con soli 6.3% di label (semi-supervised) e training significativamente più stabile dei metodi basati su contrastive learning, suggerendo che la predizione nel latente è un obiettivo di pre-training più efficiente della ricostruzione pixel o del contrastive learning esplicito.

Scaling Laws: Leggi di Potenza, Frontier Models e Limiti della Scala

La matematica della crescita dei modelli fondazionali — le leggi di Kaplan e Hoffmann, il compute-optimal frontier, i costi dell'addestramento a scala e i segnali di saturazione emergenti nei modelli di frontiera.

454.1 · Kaplan vs Hoffmann: Due Generazioni di Scaling Laws

Le **scaling laws** (Kaplan et al., 2020) stabiliscono che la loss dei LM segue relazioni di potenza con i tre fattori di scala: $L(N) \propto N^{-\alpha_N}$, $L(D) \propto D^{-\alpha_D}$, $L(C) \propto C^{-\alpha_C}$ dove N sono i parametri, D i token di training e C il compute. Kaplan et al. osservano $\alpha_N \approx 0.076$, $\alpha_D \approx 0.095$ — indicando che la scala dei parametri contribuisce più dei token di training alla riduzione della loss, portando alla strategia GPT-3 di addestrare modelli grandi su relativamente pochi token. Le **Chinchilla scaling laws** (Hoffmann et al., 2022) correggono questa analisi riaddestrando con budget compute fisso su diverse combinazioni $NimesD$: il risultato è che il rapporto ottimale è $D^* \approx 20N$ — per ogni parametro del modello, si dovrebbero usare 20 token di training. Chinchilla (70B parametri, 1.4 trilioni di token) supera GPT-3 (175B parametri, 300B token) su quasi tutti i benchmark pur usando 2.5× meno parametri, confermando che GPT-3 era massivamente underaddestrato rispetto al compute disponibile. La regola dei 20 token per parametro è diventata il thumb rule dell'industria: LLaMA (Touvron et al., 2023) addestra deliberatamente un modello da 7B su 1T token per produrre un modello efficiente all'inferenza — compute-suboptimal per Chinchilla ma ottimale per il deployment.

454.2 · Oltre Chinchilla: Inference-Optimal e i Limiti della Scala

Le leggi di Chinchilla assumono che l'obiettivo sia minimizzare la loss per un dato budget compute di training — ignorando il costo dell'inferenza. Per modelli deployati su milioni di

utenti, il costo compute di inferenza domina quello di training nell'arco di vita del modello. L'**inference-optimal scaling** (Sardana e Frankle, 2023) ridefinisce il punto ottimale: un modello più piccolo addestrando su più token (overtraining rispetto a Chinchilla) ha loss leggermente più alta ma costo di inferenza molto minore — il punto ottimale dipende dal rapporto previsto tra training e inferenza FLOP. Modelli come Gemma, Phi e LLaMA-3 seguono esplicitamente questa strategia. I segnali di **saturazione della scala** emergono su benchmark di ragionamento avanzato: MMLU e HellaSwag mostrano rendimenti decrescenti oltre i 70B parametri, mentre benchmark più difficili come MATH e GPQA mostrano ancora guadagni sostanziali — suggerendo che la scala standard dei parametri non è sufficiente per capacità di ragionamento avanzato e che test-time compute (§450) diventa il vettore di scaling dominante per la frontiera delle capacità. Il costo finanziario e energetico dell'addestramento di frontier model (GPT-4: $100M, GeminiUltra$: 200M) introduce vincoli economici che influenzano le leggi di scala operative, spostando la ricerca verso architetture più efficienti e dati di training di qualità superiore.

---

ewpage

Notazione del Volume VII

Simboli specifici introdotti nel Volume VII, in aggiunta alla notazione dei Volumi V e VI.

A.1 · Notazione per i Grandi Modelli Linguistici

Nei modelli linguistici autoregressivi, la sequenza di token è $(x_1, \ldots, x_T)$ con $x_t \in \mathcal{V}$ (vocabolario di dimensione $|\mathcal{V}|$). La probabilità condizionale del token al passo t è $P_\theta(x_t \mid x_{lt;t})$. Il scaling law di Chinchilla usa: N = numero di parametri, D = numero di token di training, $C \approx 6ND$ = budget computazionale in FLOP (approssimazione con fattore 6 per forward + backward pass). Il rapporto ottimale è $D^* \approx 20N$. Per la distillazione: T = temperatura, $\mathbf{z}_T$ = logit del teacher, $\mathbf{z}_S$ = logit dello student, $\sigma(\mathbf{z}/T)$ = soft labels del teacher alla temperatura T. Per LoRA: r = rango, $A \in \mathbb{R}^{r\times k}$, $B \in \mathbb{R}^{d\times r}$, aggiornamento $\Delta W = BA$, scaling α/r (costante solitamente uguale a r).

A.2 · Notazione per i Modelli di Diffusione

Nel processo di diffusione DDPM: T = numero di passi di diffusione; $\beta_t \in (0,1)$ = schedule di varianza al passo t; $\alpha_t = 1 - \beta_t$; $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$; $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$ = rete neurale che predice il rumore; $q(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}(\sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)I)$ = distribuzione marginale forward. Nella formulazione SDE: $\mathbf{f}(\mathbf{x}, t)$ = drift della SDE; $g(t)$ = coefficiente di diffusione; $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$ = score; $\mathbf{s}_\theta(\mathbf{x}, t) \approx \nabla_{\mathbf{x}} \log p_t(\mathbf{x})$ = score network. Per il classifier-free guidance: w = guidance scale; $\tilde{\boldsymbol{\epsilon}} = \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, \emptyset) + w(\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, \mathbf{c}) - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, \emptyset))$.

A.3 · Notazione per le GNN

Un grafo è $G = (V, E)$ con $|V| = n$ nodi e $|E| = m$ archi. La matrice di adiacenza è $\mathbf{A} \in \{0,1\}^{n \times n}$; la matrice dei gradi $\mathbf{D} = \mathrm{diag}(d_1, \ldots, d_n)$ con $d_i = \sum_j A_{ij}$. Il Laplaciano normalizzato è $\mathbf{L} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ con autovettori $\mathbf{U}$ e autovalori $0 = \lambda_1 \leq \ldots \leq \lambda_n \leq 2$. Le rappresentazioni nodali al layer k sono $\mathbf{H}^{(k)} \in \mathbb{R}^{n \times d_k}$. Il vicinato del nodo v è $\mathcal{N}(v) = \{u : (u,v) \in E\}$. Nella GIN: $\mathbf{h}_v^{(k)} = \mathrm{MLP}^{(k)}((1+\varepsilon^{(k)})\mathbf{h}_v^{(k-1)} + \sum_{u \in \mathcal{N}(v)} \mathbf{h}_u^{(k-1)})$. Per le molecole equivarianti: $\mathbf{r}_i \in \mathbb{R}^3$ sono le coordinate 3D dell'atomo i; $d_{ij} = \|\mathbf{r}_i - \mathbf{r}_j\|$ la distanza; $\hat{\mathbf{r}}_{ij} = (\mathbf{r}_i - \mathbf{r}_j)/d_{ij}$ la direzione unitaria.

Pseudocodice degli Algoritmi Principali del Volume VII

Algoritmo: Algoritmo 1 — FedAvg (Federated Averaging)

// Input: N client, E epoche locali, learning rate η, T round

Inizializza: θˆ0 sul server

per round t = 1,. . . ,T:

Seleziona sottoinsieme C [N] di client

per ogni client k ∈ C in parallelo:

$\theta_k \leftarrow$ ClientUpdate(θ, $_k$, E,η)

$\theta \hat{} 1 \leftarrow \Sigma_k \in C\ (n_k/n) \cdot \theta_k$

ClientUpdate(θ, , E, η):

per e = 1,...,E: **per** batch b $\in$: $\theta \leftarrow \theta$ - 0010$\nabla \theta$ (?)(θ) // *inner step*

$\theta \leftarrow \theta$ - 0011$\alpha_t(1-_{t-1})/(1-_t) \cdot x_t$

$x_{t-1} = \mu_t + \sqrt{\beta_t} \cdot z$

ritorna x_0

Algoritmo: Algoritmo 4 — LoRA Fine-tuning

// Input: modello pre-trained W, rango r, dati $_f t$

***Inizializza**: A (0, $\sigma \hat{} 2$), B = 0 ($\Delta W = BA = 0$ all'inizio)*

***Congela** W (non aggiornare durante il training)*

***per** ogni batch (x, y) $\in$ $_f t$:*

h = Wx + (α/r)BAx // forward con adattatore

Calcola loss (h, y)

Aggiorna solo A, B con il gradiente di

***Al deployment**: W $\leftarrow$ W + (α/r)BA // merge senza overhead*

Tabelle Comparative del Volume VII

C.1 · Confronto tra Metodi di Fine-tuning

Metodo	*Parametri addestrabili*	*Memoria GPU*	*Qualit relativa*	*Note*
Full fine-tuning	*100% ($\approx N$)*	*$\approx 4N$ bytes*	*100%*	*Baseline; ric*
LoRA ($r = 16$)	*$\sim$0.1%*	*$\approx 2N$ bytes + adattatori*	*95–99%*	*Merge a cost*
QLoRA (4-bit)	*$\sim$0.1%*	*$\approx 0.6N$ bytes*	*93–97%*	*Richiede deq*
Prefix Tuning	*$\sim$0.1%*	*$\approx 2N$ bytes*	*85–95%*	*Token virtua*
Prompt Tuning	*$\sim$0.01%*	*$\approx 2N$ bytes*	*80–92%*	*Solo input e*
Adapter	*$\sim$0.5%*	*$\approx 2N$ bytes*	*95–98%*	*Layer aggiun*

C.2 · Confronto tra Metodi di Compressione

Metodo	*Riduzione memoria*	*Riduzione latenza*	*Perdita qualit*	*Richiede training?*